안녕하세요, 전준혁입니다. 오늘도 공유의 때가 왔네요.
벌써 설이라니 믿기지가 않습니다. 간만에 1월에 설을 맞이하니 그런가봅니다.
긴 연휴동안 임직원분들을 뵙지 못할 생각에 매우 아쉽습니다. 저는 거짓말을 하지 못합니다.

그런 의미에서 설맞이 꿀팁 대방출하겠습니다.
최근 밥을 먹고 나면 왜 단 게 땡기는지 들었는데요. 음식이 들어가면 혈당량 조절을 위해 인슐린이 급격하게 늘어난다고 합니다. 그 늘어난 인슐린이 순간적으로 단 음식이 먹고 싶은 착각을 불러일으킵니다. 이때가 바로 치고 나갈 때입니다. 이 순간이 지나면 포만감 때문에 더이상 먹을 수가 없기 때문에 마지막 부스터를 가동해 가열차게 달리시기 바랍니다.
맛있는 설 음식 많이 드세요.
점심내기 윷놀이 한판 하고 싶은 날입니다.

저번에는 전기를 충전하고 방출할 수 있는 원리를 봤다면
이번에는 충전한 전기를 실제 내뿜어보는 시간입니다.

백만볼트 쏘는 피카츄입니다. 전압은 높지만 전류가 작기 때문에 살상력을 가지지 않는 착한 동물입니다.
여전히 귀엽지만 눈이 부시네요. 빠르게 스크롤을 내리도록 합니다.

파이썬으로 PCA. 주성분 분석을 해볼텐데요.
앞으로 언제나 구조가 같다는 걸 명심하시며 봐 주십시오.

1. 데이터 탐색
2. 분석을 위한 모델 선정
3. 해당 모델에 적합한 전처리
4. 모델 피팅
5. 결과값. measure 또는 metric 구하고 시각화로 표현

case마다 조금씩 다르겠지만 대략 이런 구조입니다.
항상 먼저 말씀드리고 전체 그림을 머리 속에 그리실 수 있도록 하겠습니다.

2 Class Cluster	Make	Vehicle	City MPG	Cyl	Dealer Cost	Engine Size	HP	Hwy MPG	Len	Retail Price	Weight	Wheel Base	Width
Cluster 1	Acura	Acura 3.5 RL 4dr	18	6	39,014	3.50000	225	24	197	43,755	3,880	115	72
Cluster 1	Acura	Acura 3.5 RL w/Navigation 4dr	18	6	41,100	3.50000	225	24	197	46,100	3,893	115	72
Cluster 1	Acura	Acura MDX	17	6	33,337	3.50000	265	23	189	36,945	4,451	106	77
Cluster 1	Acura	Acura NSX coupe 2dr manual S	17	6	79,978	3.20000	290	24	174	89,765	3,153	100	71
Cluster 2	Acura	Acura RSX Type S 2dr	24	4	21,761	2.00000	200	31	172	23,820	2,778	101	68
Cluster 1	Acura	Acura TL 4dr	20	6	30,299	3.20000	270	28	186	33,195	3,575	108	72
Cluster 2	Acura	Acura TSX 4dr	22	4	24,647	2.40000	200	29	183	26,990	3,230	105	69
Cluster 2	Audi	Audi A4 1.8T 4dr	22	4	23,508	1.80000	170	31	179	25,940	3,252	104	70
Cluster 2	Audi	Audi A4 3.0 4dr	20	6	28,846	3.00000	220	28	179	31,840	3,462	104	70
Cluster 2	Audi	Audi A4 3.0 convertible 2dr	20	6	38,325	3.00000	220	27	180	42,490	3,814	105	70
Cluster 2	Audi	Audi A4 3.0 Quattro 4dr auto	18	6	31,388	3.00000	220	25	179	34,480	3,627	104	70
Cluster 2	Audi	Audi A4 3.0 Quattro 4dr manual	17	6	30,366	3.00000	220	26	179	33,430	3,583	104	70
Cluster 2	Audi	Audi A4 3.0 Quattro convertible 2dr	18	6	40,075	3.00000	220	25	180	44,240	4,013	105	70
Cluster 2	Audi	Audi A41.8T convertible 2dr	23	4	32,506	1.80000	170	30	180	35,940	3,638	105	70
Cluster 1	Audi	Audi A6 2.7 Turbo Quattro 4dr	18	6	38,840	2.70000	250	25	192	42,840	3,836	109	71
Cluster 1	Audi	Audi A6 3.0 4dr	20	6	33,129	3.00000	220	27	192	36,640	3,561	109	71
Cluster 1	Audi	Audi A6 3.0 Avant Quattro	18	6	37,060	3.00000	220	25	192	40,840	4,035	109	71
Cluster 1	Audi	Audi A6 3.0 Quattro 4dr	18	6	35,992	3.00000	220	25	192	39,640	3,880	109	71
Cluster 1	Audi	Audi A6 4.2 Quattro 4dr	17	8	44,936	4.20000	300	24	193	49,690	4,024	109	71
Cluster 1	Audi	Audi A8 L Quattro 4dr	17	8	64,740	4.20000	330	24	204	69,190	4,399	121	75
Cluster 1	Audi	Audi RS 6 4dr	15	8	76,417	4.20000	450	22	191	84,600	4,024	109	78
Cluster 1	Audi	Audi S4 Avant Quattro	15	8	44,446	4.20000	340	21	179	49,090	3,936	104	70
Cluster 1	Audi	Audi S4 Quattro 4dr	14	8	43,556	4.20000	340	20	179	48,040	3,825	104	70
Cluster 2	Audi	Audi TT 1.8 convertible 2dr (coupe)	20	4	32,512	1.80000	180	28	159	35,940	3,131	95	73
Cluster 2	Audi	Audi TT 1.8 Quattro 2dr (convertible)	20	4	33,891	1.80000	225	28	159	37,390	2,921	96	73
Cluster 2	Audi	Audi TT 3.2 coupe 2dr (convertible)	21	6	36,739	3.20000	250	29	159	40,590	3,351	96	73
Cluster 2	BMW	BMW 325Ci 2dr	20	6	28,245	2.50000	184	29	177	30,795	3,197	107	69
Cluster 2	BMW	BMW 325Ci convertible 2dr	19	6	34,800	2.50000	184	27	177	37,995	3,560	107	69
Cluster 2	BMW	BMW 325i 4dr	20	6	26,155	2.50000	184	29	176	28,495	3,219	107	69
Cluster 2	BMW	BMW 325xi 4dr	19	6	27,745	2.50000	184	27	176	30,245	3,461	107	69
Cluster 2	BMW	BMW 325xi Sport	19	6	30,110	2.50000	184	26	176	32,845	3,594	107	69
Cluster 2	BMW	BMW 330Ci 2dr	20	6	33,890	3.00000	225	30	176	36,995	3,285	107	69
Cluster 2	BMW	BMW 330Ci convertible 2dr	19	6	40,530	3.00000	225	28	177	44,295	3,616	107	69
Cluster 2	BMW	BMW 330i 4dr	20	6	32,525	3.00000	225	30	176	35,495	3,285	107	69
Cluster 2	BMW	BMW 330xi 4dr	20	6	34,115	3.00000	225	29	176	37,245	3,483	107	69
Cluster 1	BMW	BMW 525i 4dr	19	6	36,620	2.50000	184	28	191	39,995	3,428	114	73
Cluster 1	BMW	BMW 530i 4dr	20	6	41,170	3.00000	225	30	191	44,995	3,472	114	73
Cluster 1	BMW	BMW 545iA 4dr	18	8	50,270	4.40000	325	26	191	54,995	3,814	114	73
Cluster 1	BMW	BMW 745i 4dr	18	8	63,190	4.40000	325	26	198	69,195	4,376	118	75
Cluster 1	BMW	BMW 745Li 4dr	18	8	66,830	4.40000	325	26	204	73,195	4,464	123	75
Cluster 1	BMW	BMW M3 convertible 2dr	16	6	51,815	3.20000	333	23	177	56,595	3,781	108	70
Cluster 1	BMW	BMW M3 coupe 2dr	16	6	44,170	3.20000	333	24	177	48,195	3,415	108	70
Cluster 1	BMW	BMW X3 3.0i	16	6	33,873	3.00000	225	23	180	37,000	4,023	110	73
Cluster 1	BMW	BMW X5 4.4i	16	8	47,720	4.40000	325	22	184	52,195	4,824	111	74
Cluster 2	BMW	BMW Z4 convertible 2.5i 2dr	20	6	31,065	2.50000	184	28	161	33,895	2,932	98	70
Cluster 2	BMW	BMW Z4 convertible 3.0i 2dr	21	6	37,575	3.00000	225	29	161	41,045	2,998	98	70
Cluster 1	Buick	Buick Century Custom 4dr	20	6	20,351	3.10000	175	30	195	22,180	3,353	109	73
Cluster 1	Buick	Buick LeSabre Custom 4dr	20	6	24,282	3.80000	205	29	200	26,470	3,567	112	74
Cluster 1	Buick	Buick LeSabre Limited 4dr	20	6	29,566	3.80000	205	29	200	32,245	3,591	112	74
Cluster 1	Buick	Buick Park Avenue 4dr	20	6	32,244	3.80000	205	29	207	35,545	3,778	114	75
Cluster 1	Buick	Buick Park Avenue Ultra 4dr	18	6	36,927	3.80000	240	28	207	40,720	3,909	114	75
Cluster 1	Buick	Buick Rainier	15	6	34,357	4.20000	275	21	193	37,895	4,600	113	75
Cluster 1	Buick	Buick Regal GS 4dr	18	6	26,047	3.80000	240	28	196	28,345	3,536	109	73
Cluster 1	Buick	Buick Regal LS 4dr	20	6	22,835	3.80000	200	30	196	24,895	3,461	109	73
Cluster 1	Buick	Buick Rendezvous CX	19	6	24,085	3.40000	185	26	187	26,545	4,024	112	74

add 편에서 봤던 Cars.csv 입니다. 차종 별 스펙이 기록돼 있고요.

```
1  import numpy as np
2  import pandas as pd
3
4
5  df = pd.read_csv('c:/users/junhyuk/downloads/TabPy materials/Cars.csv')
6  df.head()
7  pd.set_option('display.max_columns',50)
8  pd.set_option('display.width',300)
```

파이썬에서 csv파일을 불러와 읽는 겁니다.
한 줄씩 차례대로 보겠습니다. 차근차근 따라와 주세요.
궁금하실 땐 ctrl+q (quick documentation), ctrl+p (parameter info) 기억나시죠?

==========

import numpy as np
import pandas as pd

항상 쓰는 모듈을 import 했습니다.
numpy는 대수 계산 모듈이고, 앞으로 np라는 이름으로 사용할 거고요.
pandas는 dataframe 구조를 지원해줍니다. 앞으로 pd라는 이름으로 사용하겠습니다.
사용하면서 하나씩 알아가시쥬.

==========

df = pd.read_csv('c:/users/junhyuk/downloads/TabPy materials/Cars.csv')

pandas 모듈에서 read_csv 함수를 써서 csv파일을 읽어들여
dataframe 형식으로 df라는 객체에 저장했습니다.
코드를 간략히 하기 위해 그냥 파일 경로를 다 써줬습니다.
슬래쉬를 쓰고, 대소문자는 구분하지 않습니다.
헤더는? 인덱스는? 어떤 전처리를 할래? encoding은? 등의 옵션이 매우 많습니다.
나중에 필요에 따라 찾아 쓰시면 되니 걱정마세유.

==========

불러왔으니 데이터가 어떻게 생겼는지 알아보고 싶습니다.
EDA(Explotary Data Analysis. 탐색적 데이터 분석) 과정 중 일부입니다.
원래 그래프도 그리고 null, 이상치도 찾고 하는데 간단한 것들만 가져왔습니다.
세 가지만 보겠습니다.
아니 네 가지.
아니 다섯 가지만요.

```
df.head()
df.info()
df.shape
len(df)
df.describe()
```

df.head()로 df의 상위 5줄을 볼 수 있습니다. column과 index도요.
옵션으로 상하위 조절도 가능하고 5줄도 조절가능합니다.
괄호 안에 파라미터를 적지 않으면 default 옵션으로 자동 리턴해줍니다.

df.info()는 컬럼별 데이터 타입, 전체 구조, 메모리사용 등 전반적인 정보를 알려줍니다.

df.shape는 (행 수, 컬럼 수) 알려주고요.
len(df)는 행 수만 알려줍니다.
사이즈를 아는 것도 중요하지만 이걸 나중에 파라미터로 많이 쓰니까 미리 말씀드렸습니다.

df.describe()는 컬럼별 count, min, max, mean, std, 사분위수를 알려줍니다.

그림에는 안 적었지만 마음껏 실행시켜보시기 바랍니다.

==========

```
pd.set_option('display.max_columns',50)
pd.set_option('display.width',300)
```

df.head()를 적었을 때 뭔가 불편한 점이 있었지유.
화면에 내용이 끊겨서 나오는 겁니다.
그래서 display 옵션을 조정해야합니다.
단어만 봐도 사실 느낌이 오실 겁니다. 우리 영어의 수준이 이렇게 높습니다. 한국어는 얼마나 높게요.
width 의 단위는 모르겠군요. 적당히 몇 개 해보고 300이 낫겠다 싶었습니다.
해당 명령어를 실행시킨 후 다시 df.head()를 보시면 전체가 다 표현되는 걸 보실 수 있습니다.

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
```

이제 PCA를 실행할 수 있게 필요한 모듈을 import 하겠습니다.
이런건 외우실 필요 없습니다.
생김새만 보시고 다음에 필요할 때 구글링해서 찾아쓰면 됩니다.
scikit-learn(싸이킷런) 이라는 패키지에는 많은 머신러닝모델이 들어있습니다.
그 중 한 카테고리인 decomposition에서 PCA를 가져옵니다.

또 다양한 전처리를 지원하는데요.
preprocessing 에서 scaler와 encoder 하나씩 가져오겠습니다.
쓰임새는 곧 알려드리겠습니다.

```
scaler = StandardScaler()
scaled_df = scaler.fit_transform(df)
```

PCA는 반드시 scaling 해야한다고 말씀드렸습니다.
여러 가지 scaler가 있는데요. 여기서는 평균 0, 분산 1이 돼야한다고 했쥬?
그러려면 컬럼별로 mean을 빼고 std(standard deviation, 분산)을 나눠주는 작업을 해야하는데요.
이 StandardScaler()가 대신 해줍니다.
코드를 한 두 줄 줄일 수 있겠네요.

scaler를 StandardScaler로 설정하고 scaler 를 data에 fit, 그리고 transform 으로 변환합니다.
data에 fit 한다는 말은, 지금의 경우엔

해당 data에 맞는 column별 mean과 std를 구해놓는다는 말입니다.
transform 하면 data column별로 (X-mean)/std 하겠지유?

지금은 fit_transform 이 합쳐져 있어서 한 번에 됩니다. mean, std도 구하고 데이터 변환도 바로 해줍니다.
그럼 왜 나눠놨을까요?
여러가지 쓰임새가 있겠지만 대표적인 한 가지는,
train, validation, test set을 나누는 작업 때문입니다.
train set으로 학습하고 validation, test set은 새로운 data에 대해 모델의 성능을 보는 단계라 validation, test set이 없다 생각하고 학습을 진행해야합니다.
그래서 train set 으로 fit 하고 validation, test set에서는 train set에서 fit한 걸 그대로 쓰는거죠.
종종 나오는 시나리오니 case를 만나보시면 더 이해를 도와드릴 수 있겠습니다.
여유를 가지소서.
지금은 train-validation-test가 필요없는 과정이라 fit_transform 을 한 번에 하면 됩니다.

그런데 위 코드는 함정이었습니다.
오류가 날 걸요?

```
    return self.partial_fit(X, y)
  File "C:\Users\JunHyuk\my-tabpy-env\lib\site-packages\sklearn\preprocessing\_data.py",
    force_all_finite='allow-nan')
  File "C:\Users\JunHyuk\my-tabpy-env\lib\site-packages\sklearn\utils\validation.py", lin
    array = np.asarray(array, order=order, dtype=dtype)
  File "C:\Users\JunHyuk\my-tabpy-env\lib\site-packages\numpy\core\_asarray.py", line 85,
    return array(a, dtype, copy=False, order=order)
ValueError: could not convert string to float: 'Cluster 1'
```

대략 이런 내용이지유.
추측하자면, standard scaler는 평균을 빼고 분산을 나누는 계산이니 대상이 숫자여야하는데 문자가 들어왔다입니다.
생각해보니 문자는 더하기, 빼기가 안 되네요.
생각해보니 문자가 몇 컬럼 있었던 것 같습니다.

```
>>> df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 385 entries, 0 to 384
Data columns (total 15 columns):
City MPG           385 non-null int64
Cyl                385 non-null int64
Dealer Cost        385 non-null int64
Engine Size        385 non-null float64
HP                 385 non-null int64
Hwy MPG            385 non-null int64
Len                385 non-null int64
레코드 수              385 non-null int64
Retail Price       385 non-null int64
2 Class Cluster    385 non-null object
Make               385 non-null object
Vehicle            385 non-null object
Weight             385 non-null int64
Wheel Base         385 non-null int64
Width              385 non-null int64
dtypes: float64(1), int64(11), object(3)
memory usage: 45.2+ KB
```

,

df.info()를 다시 쳐 봤습니다.
총 385행에 null이 없나봅니다.
object 타입의 column 이 3 개 있네요.
각각 data가 어떤지 보겠습니다.

```
df['2 Class Cluster'].unique()
df['Make'].unique()
df.Vehicle.unique()
```

column별 유일값을 볼 수 있습니다. 한 줄씩 실행시켜 보겠습니다.
column을 부를 땐 dataframe 명에 대괄호, 따옴표 사용해서 위 그림과 같이 적어주시면 됩니다.
또는 세 번째 줄처럼 점을 찍고 column명을 쓰셔도 됩니다.

```
>>> df['2 Class Cluster'].unique()
array(['Cluster 1', 'Cluster 2'], dtype=object)
```

2 Class Cluster 라는 column 에는 두 가지 값이 있네요.

```
>>> df['Make'].unique()
array(['Acura', 'Audi', 'BMW', 'Buick', 'Cadillac', 'Chevrolet',
       'Chrvsler', 'Chrysler', 'CMC', 'Dodge', 'Ford', 'GMC', 'Honda',
       'Hummer', 'Hyundai', 'Infiniti', 'Isuzu', 'Jaguar', 'Jeep', 'Kia',
       'Land', 'Lexus', 'Lincoln', 'Mazda', 'Mazda6', 'Mercedes-Benz',
       'Mercury', 'Mini', 'Mitsubishi', 'Nissan', 'Oldsmobile', 'Pontiac',
       'Porsche', 'Saab', 'Saturn', 'Scion', 'Subaru', 'Suzuki', 'Toyota',
       'Volkswagen', 'Volvo'], dtype=object)
```

Make에는 뭐가 많습니다. 자동차 maker 같네요.

```
>>> df.Vehicle.unique()
array(['Acura 3.5 RL 4dr', 'Acura 3.5 RL w/Navigation 4dr', 'Acura MDX',
       'Acura NSX coupe 2dr manual S', 'Acura RSX Type S 2dr',
       'Acura TL 4dr', 'Acura TSX 4dr', 'Audi A4 1.8T 4dr',
       'Audi A4 3.0 4dr', 'Audi A4 3.0 convertible 2dr',
       'Audi A4 3.0 Quattro 4dr auto', 'Audi A4 3.0 Quattro 4dr manual',
       'Audi A4 3.0 Quattro convertible 2dr',
```

Vehicle은 차종입니다. 많아서 한 화면에 다 담을 수도 없네요. 아마도 차종이 ID처럼 작동할 것 같은데 확인해볼까요?

```
>>> df.Vehicle.value_counts()
Audi TT 1.8 Quattro 2dr (convertible)    1
Infiniti M45 4dr                         1
Audi A6 2.7 Turbo Quattro 4dr            1
Chrysler Sebring 4dr                     1
Kia Spectra 4dr                          1
                                        ..
Mercedes-Benz SL55 AMG 2dr               1
Mazda MX-5 Miata LS convertible 2dr      1
```

각 value마다 몇 개인지 나오는데요. 다 1개같지만 그래도 더 확실히 확인해보겠습니다.

```
>>> df.Vehicle.value_counts().sort_values(ascending=False)
Audi A6 3.0 Quattro 4dr                    1
Cadillac CTS VVT 4dr                       1
Dodge Neon SXT 4dr                         1
Lincoln LS V8 Ultimate 4dr                 1
Audi A6 3.0 Avant Quattro                  1
                                          ..
Nissan Pathfinder Armada SE                1
Toyota Camry Solara SLE V6 2dr             1
```

내림차순으로 정렬했습니다. 확실히 전부 하나네요.

그러면,

1. Make와 Vehicle은 index로 사용하고
2. 2 Class Cluster는 숫자 0, 1로 바꾸겠습니다.

categorical value 는 주로 2번처럼 0, 1로 바꾸는 작업을 진행합니다.
지금은 고유값이 두 개라서 이렇게 했지만, category가 세 개 이상이면 어떻게 할까요?
종류 개수만큼 숫자로 바꾸고 one-hot encoding 합니다. 다음에 설명드리겠습니다.
우선 위의 1, 2번을 하시쥬.

```
>>> df = df.set_index(['Make','Vehicle'])
>>> df
                                  City MPG  Cyl  Dealer Cost  Engine Size   HP  Hwy MPG  Len  레코드 수  Retail Price
Make   Vehicle
Acura  Acura 3.5 RL 4dr                 18    6        39014          3.5  225       24  197       1         43755
       Acura 3.5 RL w/Navigation 4dr    18    6        41100          3.5  225       24  197       1         46100
       Acura MDX                        17    6        33337          3.5  265       23  189       1         36945
       Acura NSX coupe 2dr manual S     17    6        79978          3.2  290       24  174       1         89765
       Acura RSX Type S 2dr             24    4        21761          2.0  200       31  172       1         23820
                                      ...  ...          ...          ...  ...      ...  ...     ...          ...
Volvo  Volvo S80 2.9 4dr                20    6        35542          2.9  208       28  190       1         37730
       Volvo S80 T6 4dr                 19    6        42573          2.9  268       26  190       1         45210
       Volvo V40                        22    4        24641          1.9  170       29  180       1         26135
       Volvo XC70                       20    5        33112          2.5  208       27  186       1         35145
```

df.set_index([‘Make’,’Vehicle’])는 dataframe의 특정 column을 index로 바꿔줍니다.

```
le = LabelEncoder()
le_df = le.fit_transform(df)
le_df = le.fit_transform(df['2 Class Cluster'])
le_df
le_df.shape
len(le_df)
df['2 Class Cluster'] = le_df  # 해도 되지만 아래 것으로 한 번에.
df['2 Class Cluster'] = le.fit_transform(df['2 Class Cluster'])  # 이거시 결론
```

이번엔 label encoder를 써야합니다.
아까 본 Cluster 1, Cluster 2를 0과 1로 바꿔줍니다.

le = LabelEncoder()

로 먼저 le라는 인코더를 만들어줍니다.
마찬가지 얘도 fit_transform 해야하는데요.

le.fit_transform(df) 하면 안 됩니다. 에러가 나죠? 또 함정에 걸리셨나요? 후후
dataframe 넣으면 알아서 categorical column 골라서 계산해주면 좋을텐데 그죠? 하지만
이렇게라도 만들어준 걸 고맙게 생각해야합니다.

le.fit_transform(df['2 Class Cluster'])

처럼 하나의 컬럼이나 1차원 array가 들어가야합니다.

le_df 결과를 보시면

```
>>> le_df
array([0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
       1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0,
       0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0,
       0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
```

이렇게 됩니다. Cluster 1, Cluster 2 가 0과 1로 바꼈네요.

```
>>> le_df.shape
(385,)
>>> len(le_df)
385
```

모양새는 이렇게 됩니다.

지금 만든 le_df를 df['2 Class Cluster']에 덮어씌워도 되지만, 가장 아래 줄을 입력해서 간단하게 한 줄로 처리하도록 합니다.

df['2 Class Cluster'] = le.fit_transform(df['2 Class Cluster'])

object들을 처리했으니 이제 scaler가 될 겁니다.
다시 실행시켜 보겠습니다.

```
>>> scaled_df = scaler.fit_transform(df)
>>> scaled_df
array([[-0.44083859,  0.16364192,  0.47832033, ...,  0.49159156,
         1.0999029 ,  0.21062934],
       [-0.44083859,  0.16364192,  0.59469839, ...,  0.50998362,
         1.0999029 ,  0.21062934],
       [-0.63068518,  0.16364192,  0.16160018, ...,  1.29942727,
        -0.16921583,  1.69583621],
       ...,
       [ 0.31854779, -1.17682915, -0.32355016, ..., -1.00523889,
        -0.87428179, -0.97753616],
       [-0.0611454 , -0.50659361,  0.14904741, ...,  0.41094947,
         0.25382375,  0.50767071],
       [-1.01037837,  0.16364192,  0.46922655, ...,  1.56398993,
```

이제 되네요. df를 scale했고 scaled_df 에 저장했습니다.
컬럼도 없이 그냥 array 로 나오는데요.
dataframe에서 직접 수식을 넣고 계산해도 됩니다. 그러면 column, index를 갖고 있는 결과가 나오겠지유? 궁금하신 분 계시면 같이 해 봅시다.

```
>>> scaled_df.shape
(385, 13)
```

모양새가 이런걸 보니 잘 된 것 같습니다.

이제 PCA 를 돌려볼 차례입니다.

```
>>> pca = PCA(n_components=5)
>>> pca_scores = pca.fit_transform(scaled_df)
>>> pca_scores
array([[-1.79981563, -0.49718732,  0.34576515,  0.39773374,  0.00533698],
       [-1.86364141, -0.38809149,  0.40101919,  0.46814958,  0.02970539],
       [-2.12101001, -0.43039306, -0.47330961, -0.0855426 ,  1.36357657],
       ...,
       [ 2.53765264,  0.42128491, -0.38353803,  0.61213958, -0.09919497],
       [-0.36067941, -0.43951561,  0.20382008,  0.32939637,  1.10374849],
       [-2.50162061, -0.3925718 , -0.75626301,  0.7066066 ,  0.98323768]])
>>> pca_scores.shape
(385, 5)
```

마찬가지 pca라는 객체에 PCA를 할당합니다.

default 값으로 하면 min(n-1, p)만큼 주성분을 만들기 때문에 13개가 될 겁니다.
PC1부터 PC13까지요.
그러면 나중에 loading vector가 (13, 13) 행렬이 나와 결과를 봤을 때 헷갈리실 것 같아
n_components=5, 그러니까 주성분을 5개만 구하도록 설정했습니다.

pca.fit_transform(scaled_df)

는 각 observation마다 score를 구해줍니다. 그래서 pca_scores에 저장했습니다. 잘했쥬?

pca_scores가 어떻게 생겼나 보시면,
저렇게 생겼습니다.
array 이고 shape는 (385, 5)네요.
385개 관측치마다 5개의 주성분, 그러니까 PC1부터 PC5까지 구했나 봅니다.

그러니까 pca_scores의
1st column이 PC1 score,
2nd column이 PC2 score, …
이런 식으로 가겠군요.

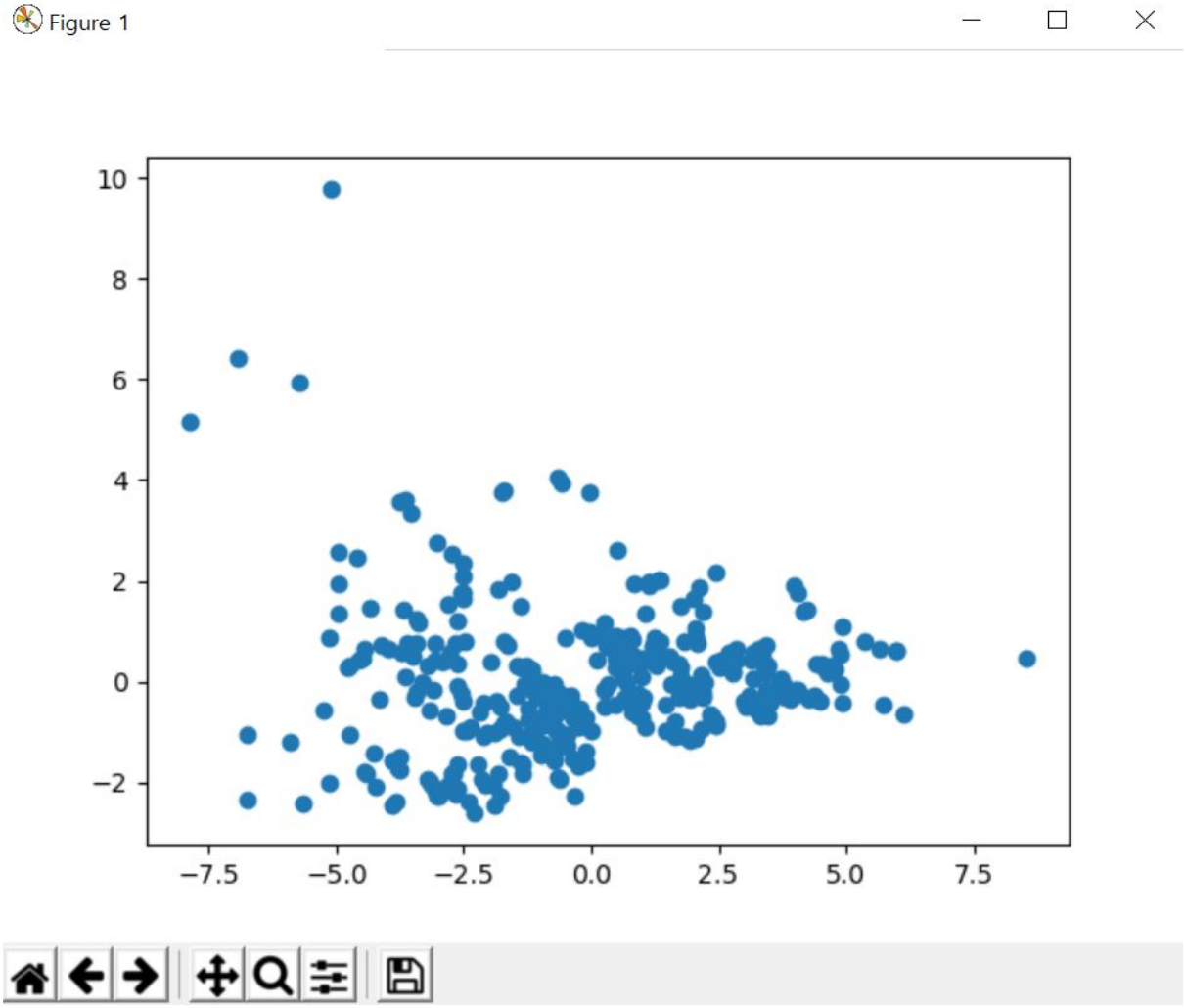
간단하게 산점도를 그려보겠습니다.
시각화는 태블로가 전문이니 여기선 간단하게만요.

```
PC1_score = pca_scores[:,0]
PC2_score = pca_scores[:,1]
```
우선 PC1, PC2 score를 저장합니다.

```
from matplotlib import pyplot as plt
plt.scatter(PC1_score, PC2_score)
```
시각화 패키지 matplotlib를 설치해야합니다.
터미널에서 pip install matplotlib 해주시구유.
그 중 pyplot을 plt라는 이름으로 쓰겠습니다.
구조는

plt.scatter(x축 값들, y축 값들, 옵션들)

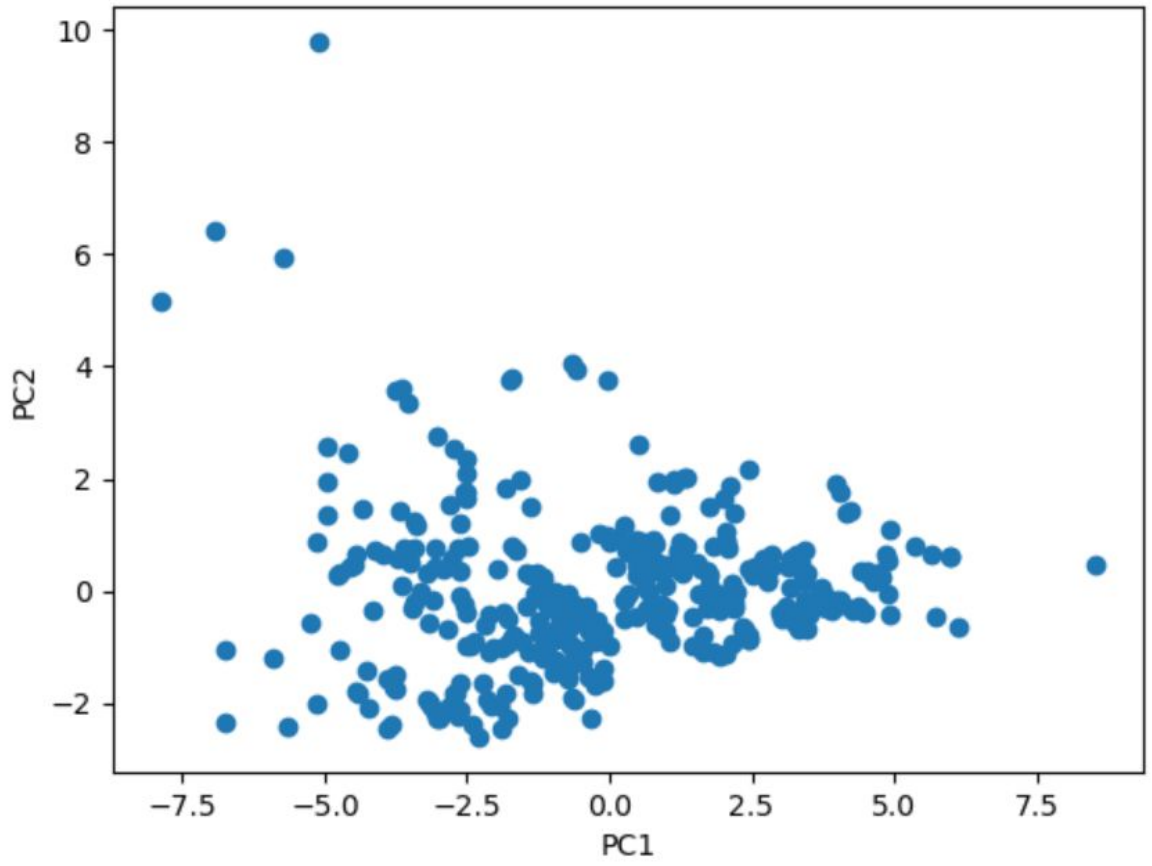
입니다.



넘나 대충 그렸다구유? 부끄럽네유.

축 이름 정도는 있으면 좋겠네요.

```
plt.xlabel('PC1')
plt.ylabel('PC2')
```

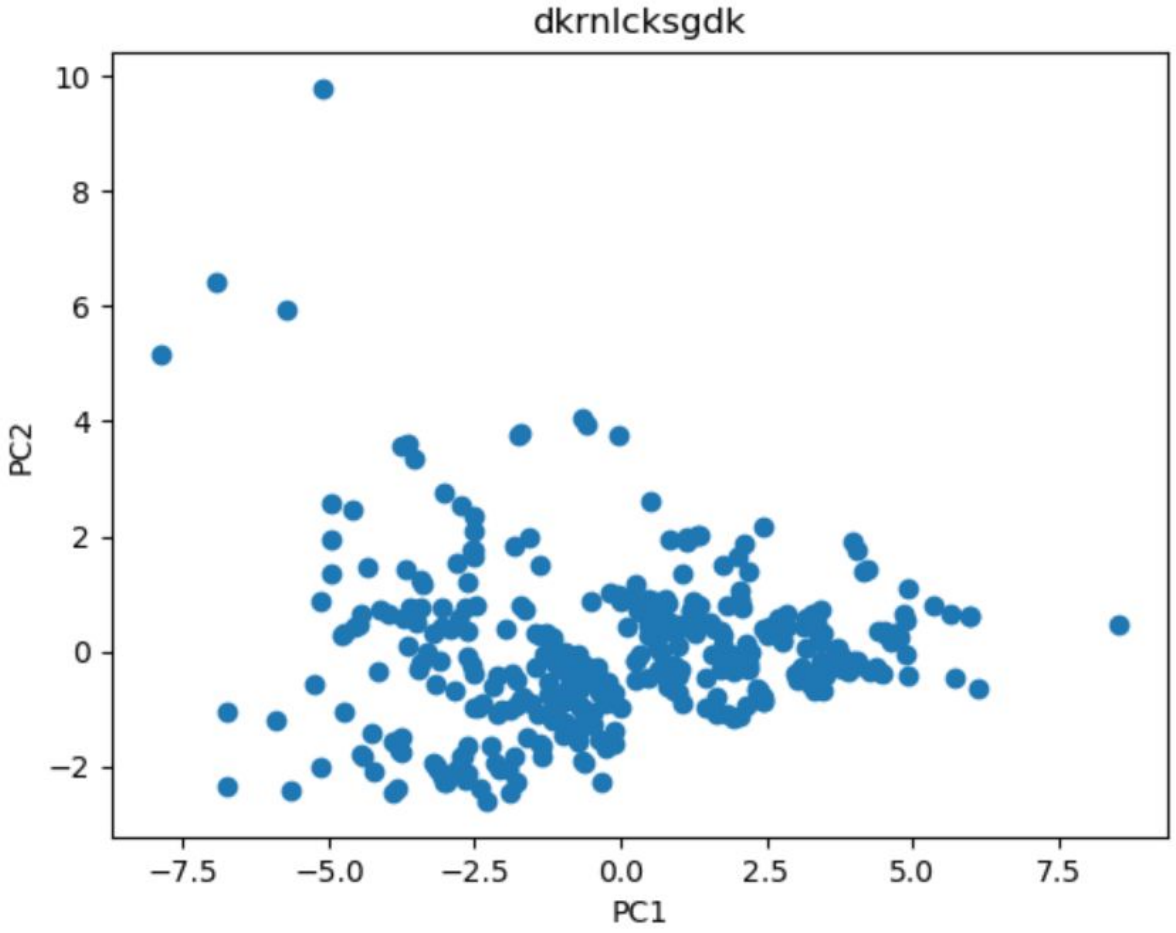


하면 이렇게 됩니다.

훨씬 낫쥬? 제목도 한번 넣어볼까여.

```
plt.title('dkrnlcksgdk')
```

요렇게 하면



요렇게 됩니다.

이전 단계에서 우리가 봤던 그림과는 다소 차이가 있습니다.

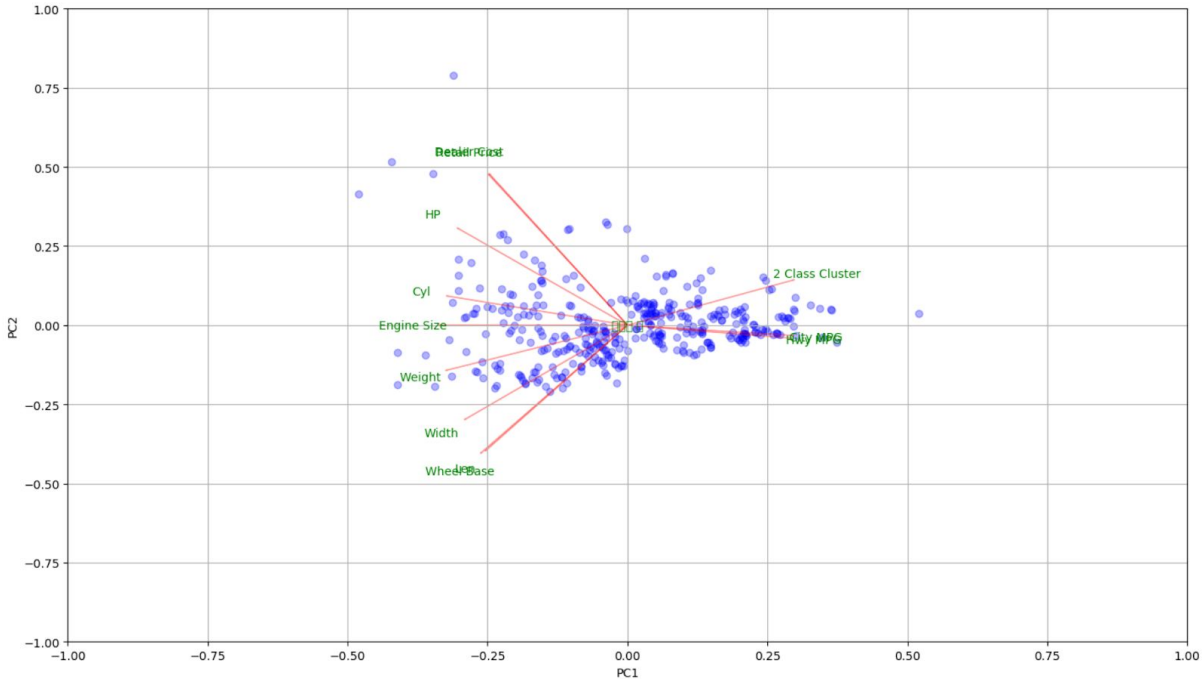
컬럼별로 loading vector 도 그려져있고 막 그랬잖아요?

그걸 한 번 만들어볼까요.

```
def myplot(score, loading, labels=None):
    PC1s = score[:, 0]
    PC2s = score[:, 1]
    n = loading.shape[0]
    scalex = 1.0 / (PC1s.max() - PC1s.min())
    scaley = 1.0 / (PC2s.max() - PC2s.min())
    plt.scatter(PC1s * scalex, PC2s * scaley, c='b',alpha=0.3)
    for i in range(n):
        plt.arrow(0, 0, loading[i, 0], loading[i, 1], color='r', alpha=0.3)
        if labels is None:
            plt.text(loading[i, 0] * 1.15, loading[i, 1] * 1.15,
                     "Var" + str(i + 1), color='g', ha='center', va='center')
        else:
            plt.text(loading[i, 0] * 1.15, loading[i, 1] * 1.15,
                     labels[i], color='g', ha='center', va='center')
    plt.xlim(-1, 1)
    plt.ylim(-1, 1)
    plt.xlabel("PC{}".format(1))
    plt.ylabel("PC{}".format(2))
    plt.grid()


myplot(pca_scores[:, 0:2], np.transpose(pca.components_[0:2, :]), df.columns)
```

갑자기 복잡해진 느낌이지만 그냥 복붙하시겠어요?
복사가 안 된다구요?
아이고 저런.



확대해서 보시면 빨간 선에 화살표가 있긴 한데... 너무 작쥬?

우리에겐 태블로가 있으니 너무 집착하지 않도록 하겠습니다.

아직 체크해보지 않은 게 있습니다.
바로 loading vector 구하는 방법과 PVE 구하고 scree plot 그리기 입니다.

pca.components_

를 치시면 loading vector를 구할 수 있습니다.

```
>>> pca.components_.shape
(5, 13)
```

shape를 보시면 5 X 13 행렬이네요.

1행이 PC1 의 loading vector
2행이 PC2 의 loading vector
…
이런 형태입니다.

이 loading vector 좌표로 위에서 빨간 화살표를 그릴 수 있었습니다.

```
>>> pca.explained_variance_ratio_
array([0.64580558, 0.16006843, 0.07366893, 0.0323605 , 0.02338144])
>>> pca.explained_variance_ratio_.sum()
0.9352848828098594
```

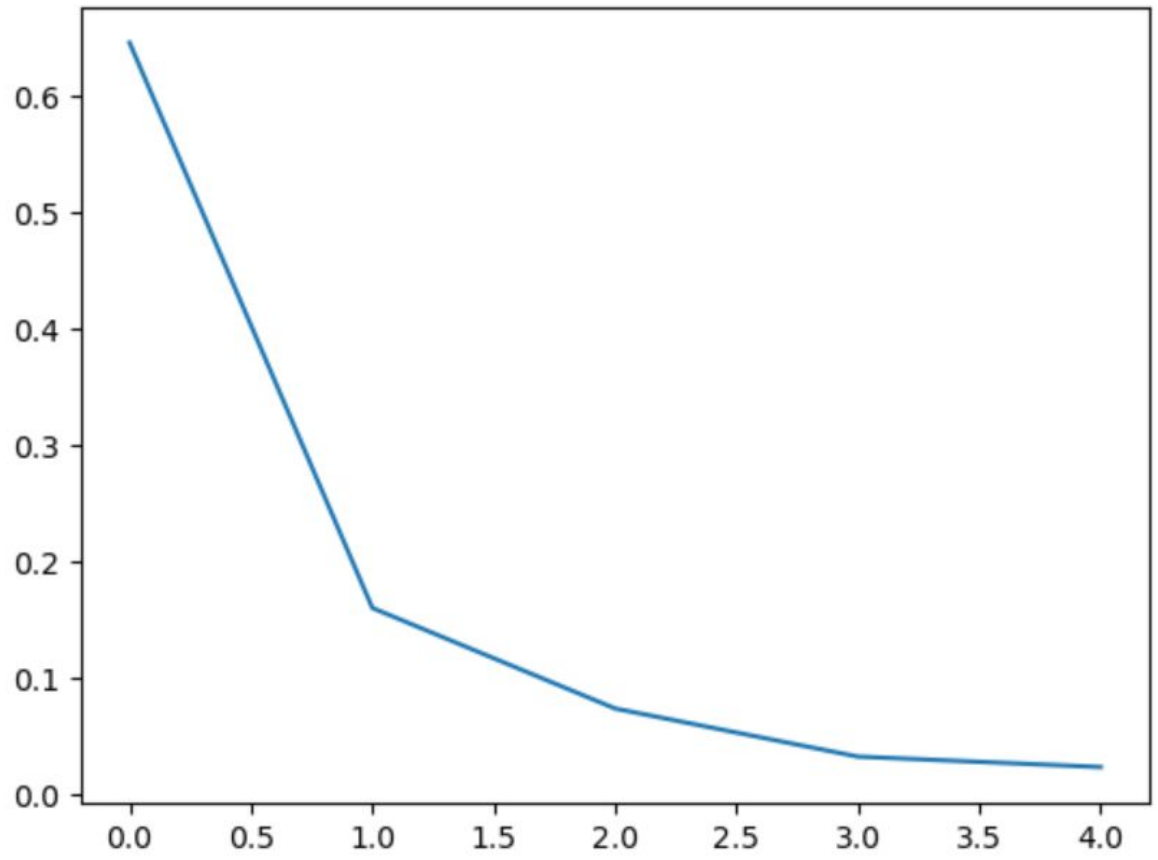
다음은 PVE 입니다.

pca.explained_variance_ratio_

로 각 주성분별 PVE를 얻을 수 있는데요.
PC1이 64%,
PC2가 16%,
PC3가 7%
…
만큼 원래 데이터를 설명해주고 있네요.
이를 scree plot으로 나타내면

```
plt.plot(pca.explained_variance_ratio_)
```

위와 같은 명령어로



요렇게 간단하게 볼 수 있습니다.
전혀 디자인이라고는 신경쓰지 않는 개발자의 일상을 따라해봤습니다.

X축이 0일 때 주성분 한 개, 1일 때 두 개, 이런 식으로 다섯 개까지 나타낸 거고요.
Y축이 PVE를 나타낸 그래프입니다.
꺾이는 부분을 보고 적합한 주성분 개수를 찾아본다고 했지유?
지금은 1에서 꺾이니 주성분을 두 개, 그러니까 PC1, PC2 까지만 하면 충분하겠습니다.

구글 워드가 15장쯤 되면 급격히 느려지네요. 구글 실망..

대시보드는 다음 문서에서 다루겠습니다.

이제 얼마남지 않았으니 힘내세요, 전준혁 선생님.

1. 데이터 탐색
2. 분석을 위한 모델 선정
3. 해당 모델에 적합한 전처리
4. 모델 피팅
5. 결과값. measure 또는 metric 구하고 시각화로 표현

이 틀이 기억나시나요? 수미상관 구조를 위해 넣었습니다.

지금은 모델 선정 과정을 넣진 않았지만 다양한 모델을 습득하고 나면 필요할 겁니다.

각 과정에 따라 뭘 했었는지 떠올려보시며 마무리하시면 좋겠습니다.

감사합니다.