**멀티캠퍼스 융복합 AI과정**

**ML 수행평가**

**이동규**

**01. ML EXCERCISE01\_movie**

# 1. 사용자가 평가한 모든 영화의 전체 평균 평점

코드

df\_ratings['rating'].mean()

결과

3.501556983616962

# 2. 각 사용자별 평균 평점

코드

pd.DataFrame(df\_ratings.groupby(['userId'], as\_index=True).mean()['rating'])

결과

userId rating

1 4.366379

2 3.948276

3 2.435897

4 3.555556

5 3.636364

... ...

# 3. 각 영화별 평균 평점

코드

s1 = df\_rating['rating'].groupby(df\_rating['movieId']).mean()

result = pd.merge(s1, df\_movies, left\_index=True, right\_on='movieId', how='inner')

display(result[['movieId', 'title', 'rating']])

결과

|  | **movieId** | **title** | **rating** |
| --- | --- | --- | --- |
| **0** | 1 | Toy Story (1995) | 3.920930 |
| **1** | 2 | Jumanji (1995) | 3.431818 |
| **2** | 3 | Grumpier Old Men (1995) | 3.259615 |
| **3** | 4 | Waiting to Exhale (1995) | 2.357143 |
| **4** | 5 | Father of the Bride Part II (1995) | 3.071429 |
| **...** | ... | ... | ... |

# 4. 평균 평점이 가장 높은 영화의 제목(동률이 있을 경우 모두 출력)

코드

s1 = df\_rating['rating'].groupby(df\_rating['movieId']).mean()

high\_rating = s1[s1 == s1.max()]

result = pd.merge(high\_rating, df\_movies, left\_index=True, right\_on='movieId', how='inner')

display(result.sort\_values(by='title')[['movieId','title','rating']])

결과

| **movieId** | **title** | **rating** |
| --- | --- | --- |
| **5690** | 27751 | 'Salem's Lot (2004) | 5.0 |
| **7332** | 77846 | 12 Angry Men (1997) | 5.0 |
| **9046** | 141816 | 12 Chairs (1976) | 5.0 |
| **3893** | 5468 | 20 Million Miles to Earth (1957) | 5.0 |
| **5639** | 27373 | 61\* (2001) | 5.0 |
| **...** | ... | ... | ... |

# 5. Comedy영화 중 가장 평점이 낮은 영화의 제목

코드

merged\_df = pd.merge(group\_rating, df\_movies, left\_index=True, right\_on='movieId', how='inner')

is\_comedy = merged\_df['genres'].str.contains('Comedy')

df\_comedy = merged\_df[is\_comedy]

df\_comedy[df\_comedy['rating'] == df\_comedy['rating'].min()]

결과

|  |  |  | **rating** | **movieId** | **title** | **genres** |
| --- | --- | --- | --- | --- | --- | --- |
| **0** |  |  | 0.5 | 1 | Toy Story (1995) | Adventure|Animation|Children|Comedy|Fantasy |
| **2** |  |  | 0.5 | 3 | Grumpier Old Men (1995) | Comedy|Romance |
| **4** |  |  | 0.5 | 5 | Father of the Bride Part II (1995) | Comedy |
| **35** |  |  | 0.5 | 39 | Clueless (1995) | Comedy|Romance |
| **56** |  |  | 0.5 | 63 | Don't Be a Menace to South Central While Drink... | Comedy|Crime |
| **...** |  |  | ... | ... | ... | ... |
| **9297** |  |  | 0.5 | 158872 | Sausage Party (2016) | Animation|Comedy |
| **9419** |  |  | 0.5 | 165645 | Bad Santa 2 (2016) | Comedy |
| **9430** |  |  | 0.5 | 166461 | Moana (2016) | Adventure|Animation|Children|Comedy|Fantasy |
| **9535** |  |  | 0.5 | 172547 | Despicable Me 3 (2017) | Adventure|Animation|Children|Comedy |
| **9590** |  |  | 0.5 | 175475 | The Emoji Movie (2017) | Animation|Children|Comedy |

488 rows × 4 columns

# 6. 2015년도에 평가된 모든 Romance 영화의 평균 평점은?

is\_Romance = df['genres'].str.contains('Romance')

df\_Romance = df[is\_Romance]

**02. ML EXCERCISE01\_mpg**

# 1. displ(배기량)이 4 이하인 자동차와 5 이상인 자동차 중

# 어떤 자동차의 hwy(고속도로 연비)가 평균적으로 더 높은지 확인하세요.

코드

my\_displ\_under = df\_mpg['displ'] <= 4.0

my\_displ\_over = df\_mpg['displ'] >= 5.0

df\_mpg[my\_displ\_over]

print(df\_mpg[my\_displ\_over].mean()['hwy'])

print(df\_mpg[my\_displ\_under].mean()['hwy'])

결과

18.07894736842105

25.96319018404908

# 2. 자동차 제조 회사에 따라 도시 연비가 다른지 알아보려고 한다.

# "audi"와 "toyota" 중 어느 manufacturer(제조회사)의 cty(도시 연비)가

# 평균적으로 더 높은지 확인하세요.

코드

man\_cty\_mean = pd.DataFrame(df\_mpg.groupby(['manufacturer']).mean()['cty'])

print(man\_cty\_mean.loc['audi'])

print(man\_cty\_mean.loc['toyota'])

결과

cty 17.611111

Name: audi, dtype: float64

cty 18.529412

Name: toyota, dtype: float64

# 3. "chevrolet", "ford", "honda" 자동차의 고속도로 연비 평균을 알아보려고 한다.

# 이 회사들의 데이터를 추출한 후 hwy(고속도로 연비) 평균을 구하세요.

코드

df\_fch = df\_mpg[(df\_mpg['manufacturer'] == 'ford') | (df\_mpg['manufacturer'] == 'honda') | (df\_mpg['manufacturer'] == 'chevrolet')]

df\_fch.mean().hwy

결과

22.50943396226415

# 4. "audi"에서 생산한 자동차 중에 어떤 자동차 모델의 hwy(고속도로 연비)가

# 높은지 알아보려고 한다. "audi"에서 생산한 자동차 중 hwy가 1~5위에 해당하는

# 자동차의 데이터를 출력하세요.

코드

df\_audi = df\_mpg[(df\_mpg['manufacturer'] == 'audi')]

df\_audi.sort\_values(['hwy']).head(5)

결과

|  | **manufacturer** | **model** | **displ** | **year** | **cyl** | **trans** | **drv** | **cty** | **hwy** | **fl** | **class** | **mean\_y** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **17** | audi | a6 quattro | 4.2 | 2008 | 8 | auto(s6) | 4 | 16 | 23 | p | midsize | 19.5 |
| **15** | audi | a6 quattro | 2.8 | 1999 | 6 | auto(l5) | 4 | 15 | 24 | p | midsize | 19.5 |
| **14** | audi | a4 quattro | 3.1 | 2008 | 6 | manual(m6) | 4 | 15 | 25 | p | compact | 20.0 |
| **13** | audi | a4 quattro | 3.1 | 2008 | 6 | auto(s6) | 4 | 17 | 25 | p | compact | 21.0 |
| **12** | audi | a4 quattro | 2.8 | 1999 | 6 | manual(m5) | 4 | 17 | 25 | p | compact | 21.0 |

# 5. mpg 데이터는 연비를 나타내는 변수가 2개입니다.

# 두 변수를 각각 활용하는 대신 하나의 통합 연비 변수를 만들어 사용하려 합니다.

# 평균 연비 변수는 두 연비(고속도로와 도시)의 평균을 이용합니다.

# 회사별로 "suv" 자동차의 평균 연비를 구한후 내림차순으로 정렬한 후 1~5위까지 데이터를 출력하세요.

코드

df\_mpg['mean\_y'] =(df\_mpg['cty'] + df\_mpg['hwy'])/2

df\_mpg[df\_mpg['class'] == 'suv'].sort\_values(['mean\_y'], ascending = False).head(5)

결과

|  | **manufacturer** | **model** | **displ** | **year** | **cyl** | **trans** | **drv** | **cty** | **hwy** | **fl** | **class** | **mean\_y** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **161** | subaru | forester awd | 2.5 | 2008 | 4 | manual(m5) | 4 | 20 | 27 | r | suv | 23.5 |
| **163** | subaru | forester awd | 2.5 | 2008 | 4 | auto(l4) | 4 | 20 | 26 | r | suv | 23.0 |
| **162** | subaru | forester awd | 2.5 | 2008 | 4 | manual(m5) | 4 | 19 | 25 | p | suv | 22.0 |
| **159** | subaru | forester awd | 2.5 | 1999 | 4 | manual(m5) | 4 | 18 | 25 | r | suv | 21.5 |
| **160** | subaru | forester awd | 2.5 | 1999 | 4 | auto(l4) | 4 | 18 | 24 | r | suv | 21.0 |

# 6. mpg 데이터의 class는 "suv", "compact" 등 자동차의 특징에 따라

# 일곱 종류로 분류한 변수입니다. 어떤 차종의 도시 연비가 높은지 비교하려 합니다.

# class별 cty 평균을 구하고 cty 평균이 높은 순으로 정렬해 출력하세요.

코드

df\_mpg.groupby(['class'],).mean().sort\_values(['cty'], ascending = False).cty

결과

class

subcompact 20.371429

compact 20.127660

midsize 18.756098

minivan 15.818182

2seater 15.400000

suv 13.500000

pickup 13.000000

Name: cty, dtype: float64

# 7. 어떤 회사 자동차의 hwy(고속도로 연비)가 가장 높은지 알아보려 합니다.

# hwy(고속도로 연비) 평균이 가장 높은 회사 세 곳을 출력하세요.

코드

df\_mpg.groupby(['manufacturer']).mean().sort\_values(['hwy'], ascending = False).head(3).hwy

결과

manufacturer

honda 32.555556

volkswagen 29.222222

hyundai 26.857143

Name: hwy, dtype: float64

# 8. 어떤 회사에서 "compact" 차종을 가장 많이 생산하는지 알아보려고 합니다.

# 각 회사별 "compact" 차종 수를 내림차순으로 정렬해 출력하세요.

코드

df\_compact = df\_mpg[df\_mpg['class'] == 'compact']

df\_compact.groupby(['manufacturer']).count().sort\_values(['model'], ascending = False).model

결과

manufacturer

audi 15

volkswagen 14

toyota 12

subaru 4

nissan 2

Name: model, dtype: int64

**03. ML EXCERCISE01\_Koweps**

# 1. 성별에 따른 월급 차이

# 과거에 비해 여성의 사회 진출이 활발하지만 직장에서의

# 위상에서는 여전히 차별이 존재하고 있는것이 사실.

# 실제로 그러한지 월급의 차이를 이용하여 사실을 확인해보자

코드

df\_sex = df[['h10\_g3', 'p1002\_8aq1']]

df\_valid = df\_sex[df\_sex['p1002\_8aq1'].notnull()]

df\_valid.groupby(['h10\_g3']).mean()

결과

|  | **p1002\_8aq1** |
| --- | --- |
| **h10\_g3** |  |
| **1.0** | 312.293165 |
| **2.0** | 162.199670 |

# 2. 나이와 월급의 관계

# 몇 살 때 월급을 가장 많이 받을까? 또 그때의 월급은 얼마인가?

코드

df\_age = df[['h10\_g4', 'p1002\_8aq1']]

df\_valid = df\_age[df\_age['p1002\_8aq1'].notnull()]

df\_valid.groupby(['h10\_g4']).mean().sort\_values(['p1002\_8aq1'], ascending = False).head(5)

결과

|  | **p1002\_8aq1** |
| --- | --- |
| **h10\_g4** |  |
| **1963.0** | 318.677748 |
| **1967.0** | 315.033810 |
| **1972.0** | 314.897324 |
| **1966.0** | 314.052143 |
| **1976.0** | 306.467203 |

# 3. 연령대에 따른 월급 차이

# 30세 미만을 초년(young),

# 30~59세 : 중년(middle),

# 60세 이상 : 노년(old)

# 위의 범주로 연령대에 따른 월급의 차이를 알아보자

코드

df['age\_cal'] = 2020-df['h10\_g4']

df\_age = df[['h10\_g4','age\_cal','p1002\_8aq1']]

df\_valid = df\_age[df\_age['p1002\_8aq1'].notnull()]

young = df\_valid['age\_cal'] <30

middle = (df\_valid['age\_cal'] >=30) & (df\_valid['age\_cal'] < 60)

old = df\_valid['age\_cal'] >= 60

print('young' , df\_valid[young].mean().p1002\_8aq1)

print('middle' ,df\_valid[middle].mean().p1002\_8aq1)

print('old' ,df\_valid[old].mean().p1002\_8aq1)

결과

young 134.2502830188679

middle 275.5161802184467

old 162.61112788632326

# 4. 연령대 및 성별 월급 차이

# 성별 월급 차이는 연령대에 따라 다른 양상을 보일 수 있습니다.

# 성별 월급 차이가 연령대에 따라 다른지 분석해보자

# 기존에는 3그룹(초년,중년,노년)이었지만 이젠 6그룹으로

# 그룹핑을 해야 한다.(초년남성,초년여성,..)

코드

df['age\_cal'] = 2020-df['h10\_g4']

df\_age\_sex = df[['h10\_g3','h10\_g4','age\_cal','p1002\_8aq1']]

df\_valid = df\_age\_sex[df\_age\_sex['p1002\_8aq1'].notnull()]

young\_m = (df\_valid['age\_cal'] <30) & (df\_valid['h10\_g3'] ==1)

young\_f = (df\_valid['age\_cal'] <30) & (df\_valid['h10\_g3'] ==2)

middle\_m = (df\_valid['age\_cal'] >=30) & (df\_valid['age\_cal'] < 60) & (df\_valid['h10\_g3'] ==1)

middle\_f = (df\_valid['age\_cal'] >=30) & (df\_valid['age\_cal'] < 60) & (df\_valid['h10\_g3'] ==2)

old\_m = (df\_valid['age\_cal'] >= 60) & (df\_valid['h10\_g3'] ==1)

old\_f = (df\_valid['age\_cal'] >= 60) & (df\_valid['h10\_g3'] ==2)

df\_valid['group'] = 0

df\_valid['group'][young\_m] = 'young\_m'

df\_valid['group'][young\_f] = 'young\_f'

df\_valid['group'][middle\_m] = 'middle\_m'

df\_valid['group'][middle\_f] = 'middle\_f'

df\_valid['group'][old\_m] = 'old\_m'

df\_valid['group'][old\_f] = 'old\_f'

df\_valid.groupby(['group']).mean()

결과

Out[85]:

|  |  | **h10\_g3** | **h10\_g4** | **age\_cal** | **p1002\_8aq1** |
| --- | --- | --- | --- | --- | --- |
| **group** |  |  |  |  |  |
| **middle\_f** |  | 2.0 | 1974.590878 | 45.409122 | 190.379735 |
| **middle\_m** |  | 1.0 | 1974.220033 | 45.779967 | 343.970170 |
| **old\_f** |  | 2.0 | 1950.310954 | 69.689046 | 96.322703 |
| **old\_m** |  | 1.0 | 1951.807143 | 68.192857 | 229.609786 |
| **young\_f** |  | 2.0 | 1992.462585 | 27.537415 | 134.239456 |
| **young\_m** |  | 1.0 | 1992.538462 | 27.461538 | 134.274769 |

# 5. 직업별 월급 차이

# 어떤 직업이 월급을 가장 많이 받을까?

# 직업별 월급을 분석해 보자

# 직업코드는 제공된 Koweps\_Codebook.xlsx을 이용하면

# 편하게 코드값을 이용 할 수 있습니다.

코드

df\_valid = df[['h10\_eco9','p1002\_8aq1']][df['p1002\_8aq1'].notnull()]

df\_valid.groupby(['h10\_eco9']).mean().sort\_values(['h10\_eco9']).head(5)

결과

|  | **p1002\_8aq1** |
| --- | --- |
| **h10\_eco9** |  |
| **111.0** | 750.000000 |
| **120.0** | 563.763333 |
| **131.0** | 549.912500 |
| **132.0** | 726.180000 |
| **133.0** | 375.446000 |

# 6. 성별 직업 빈도

# 성별로 어떤 직업이 가장 많을까?

코드

df\_sex\_job = df[['h10\_g3','h10\_eco9']]

df\_m\_job = df\_sex\_job[df\_sex\_job['h10\_g3'] == 1].groupby(['h10\_eco9']).count()

display(df\_m\_job.sort\_values(['h10\_g3'],ascending=False).head(3))

df\_f\_job = df\_sex\_job[df\_sex\_job['h10\_g3'] == 2].groupby(['h10\_eco9']).count()

display(df\_f\_job.sort\_values(['h10\_g3'],ascending=False).head(3))

결과

|  | **h10\_g3** |
| --- | --- |
| **h10\_eco9** |  |
| **611.0** | 640 |
| **873.0** | 251 |
| **312.0** | 213 |
|  | **h10\_g3** |
| **h10\_eco9** |  |
| **611.0** | 680 |
| **941.0** | 228 |
| **521.0** | 221 |

# 7. 종교 유무에 따른 이혼율

# 종교가 있는 사람들이 이혼을 덜 할까??

코드

#h10\_g10 혼인, 3 이혼

#h10\_g11 1 있음

df\_mar\_rel = df[['h10\_g10','h10\_g11']]

# 종교가 있는 사람들

df\_rel\_y = df\_mar\_rel['h10\_g11'] == 1

df\_rel\_y = df\_mar\_rel[df\_rel\_y]

# 결혼유무 그룹핑

df\_rel\_y\_sum = df\_rel\_y.groupby(['h10\_g10']).sum()

print('종교 유, 이혼율:', df\_rel\_y\_sum[3:3]['h10\_g11'].sum()/ df\_rel\_y\_sum[0:]['h10\_g11'].sum())

# 종교가 없는 사람들

df\_rel\_n = df\_mar\_rel['h10\_g11'] == 2

df\_rel\_n = df\_mar\_rel[df\_rel\_n]

# 결혼유무 그룹핑

df\_rel\_n\_sum = df\_rel\_n.groupby(['h10\_g10']).sum()

print('종교 무, 이혼율:', df\_rel\_n\_sum[3:3]['h10\_g11'].sum()/ df\_rel\_n\_sum[0:]['h10\_g11'].sum())

결과

종교 유, 이혼율: 0.04076053187523301

종교 무, 이혼율: 0.04456307299524196

# 8. 지역별 연령대 비율

# 노년층이 많은 지역은 어디일까?

코드

df['age\_cal'] = 2020-df['h10\_g4']

df\_age = df[['h10\_g4','age\_cal','h10\_reg7']]

df\_valid = df\_age[df\_age\_sex['p1002\_8aq1'].notnull()]

old = (df\_valid['age\_cal'] >= 60)

df\_valid[old].groupby(['h10\_reg7']).count().sort\_values(['h10\_g4'],ascending =False).h10\_g4

결과

h10\_reg7

2.0 283

1.0 201

3.0 198

7.0 160

4.0 123

6.0 92

5.0 69

Name: h10\_g4, dtype: int64

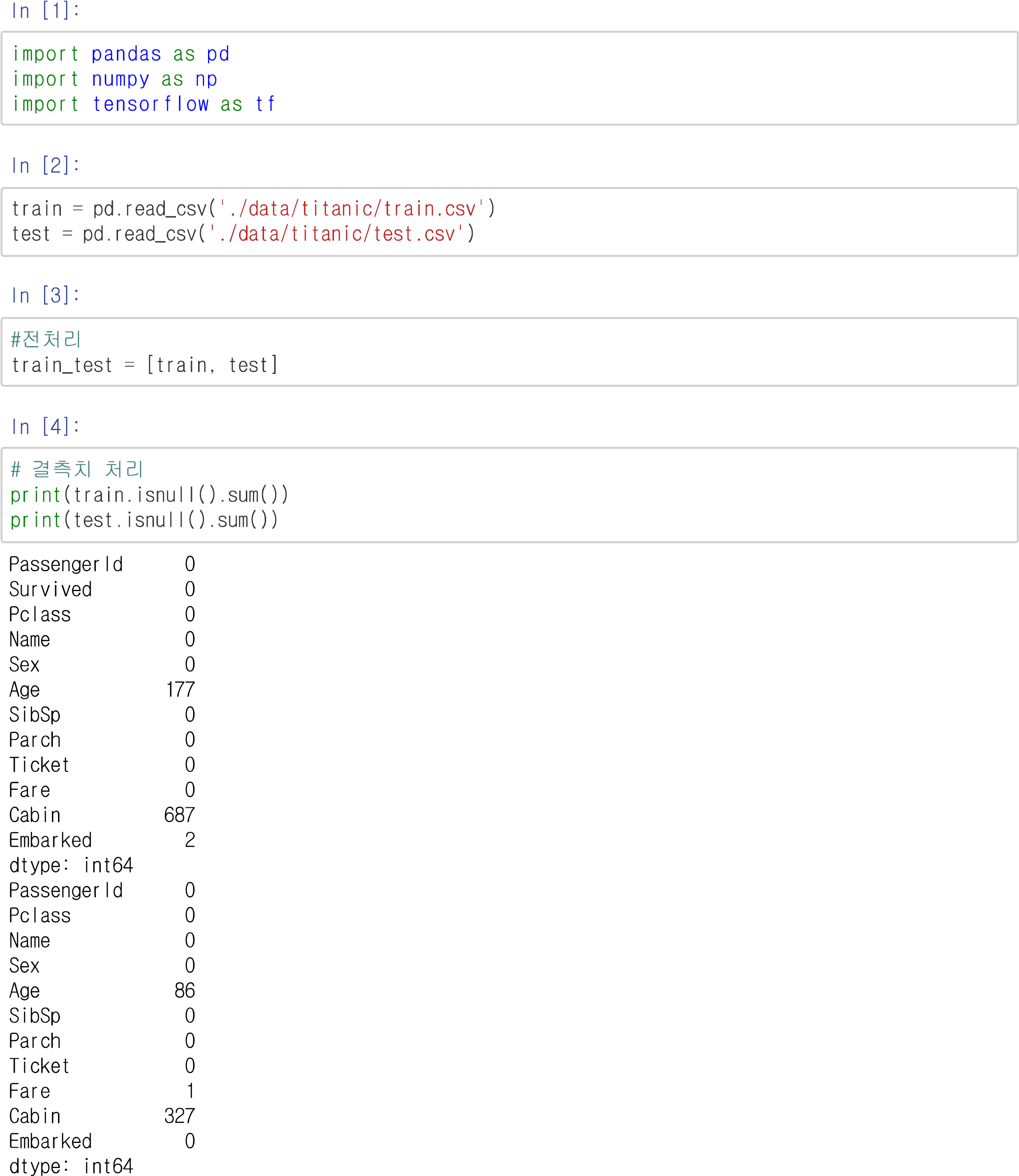
**04. ML EXCERCISE02\_Titanic**

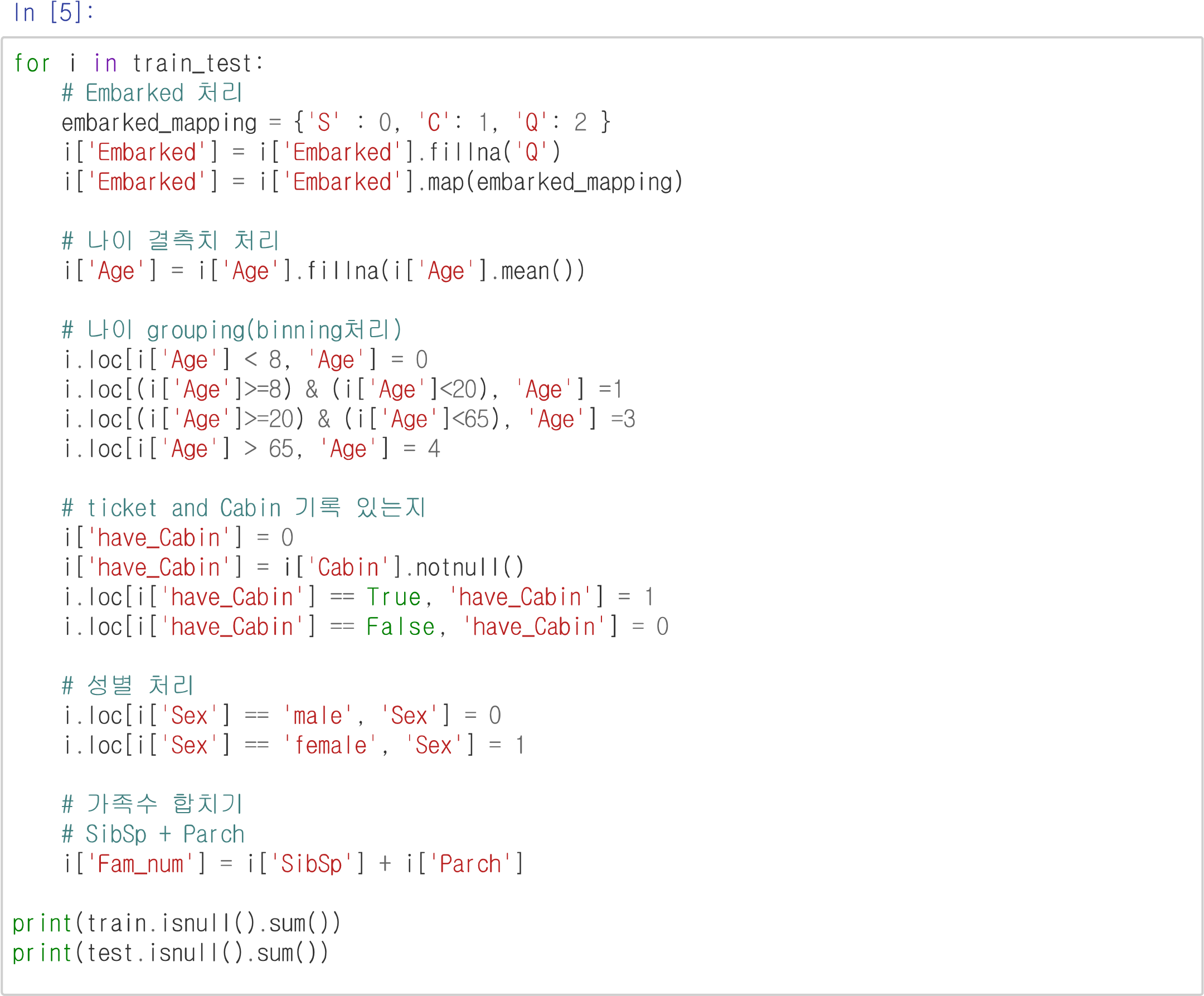
**결과**

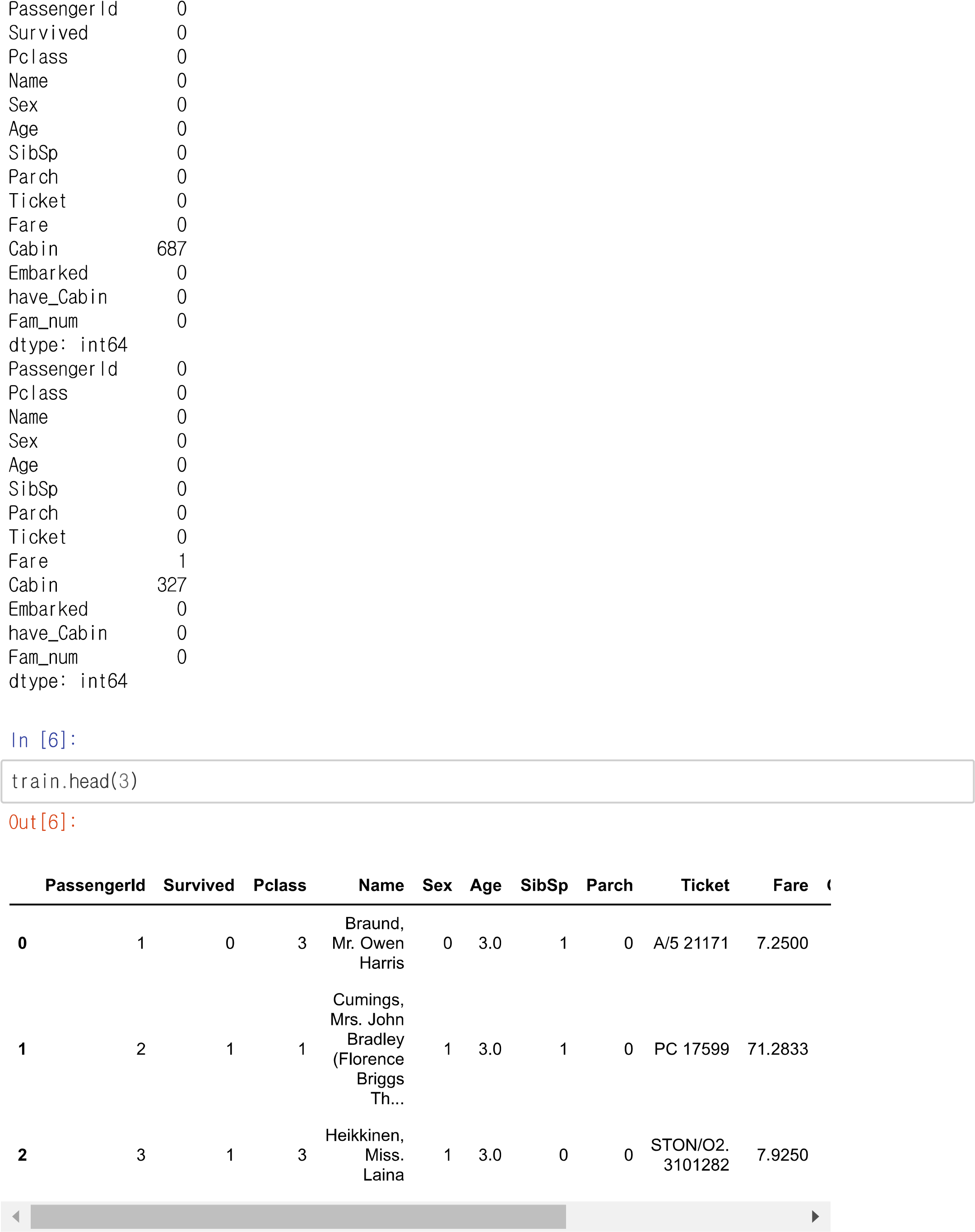
**스크린샷, 모니터, 화면, 컴퓨터이(가) 표시된 사진

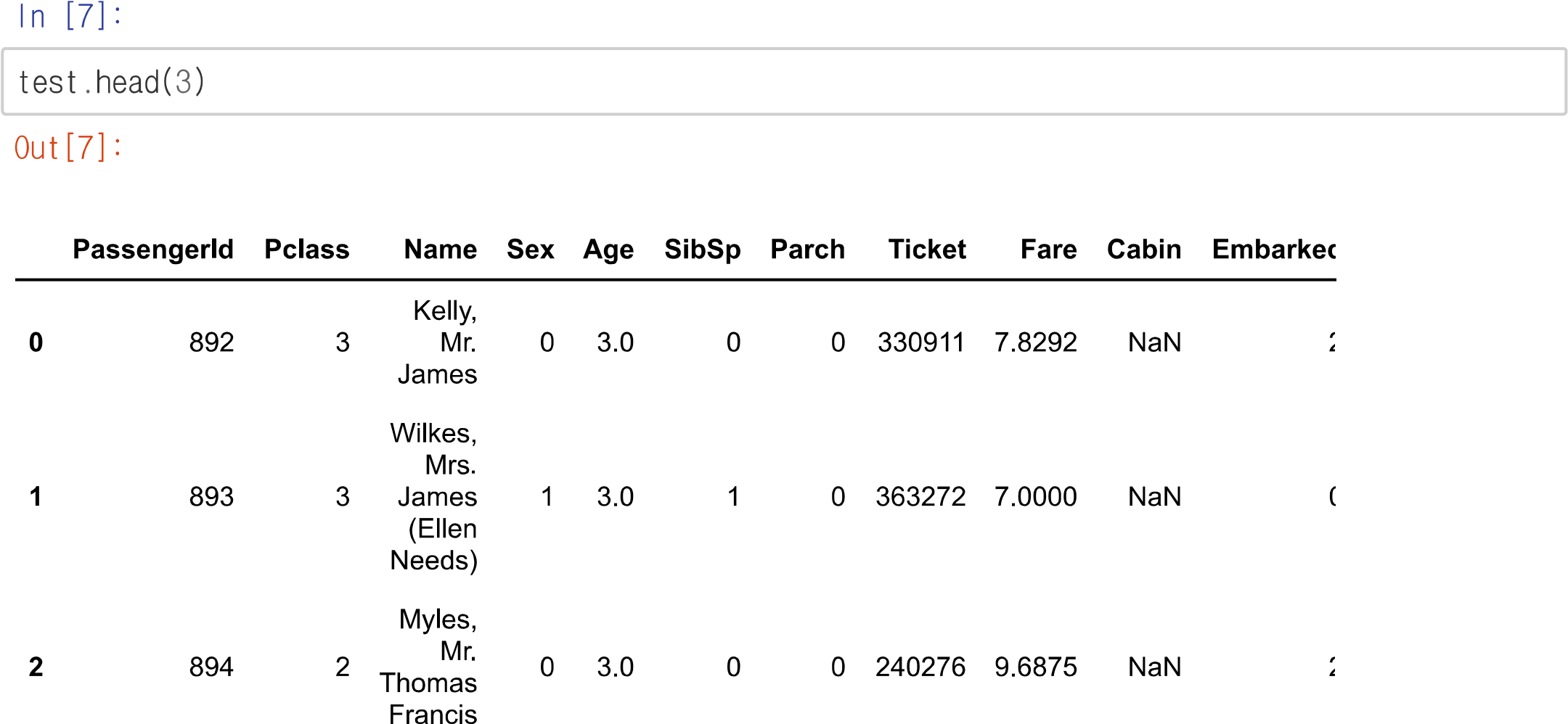
자동 생성된 설명**

**코드**



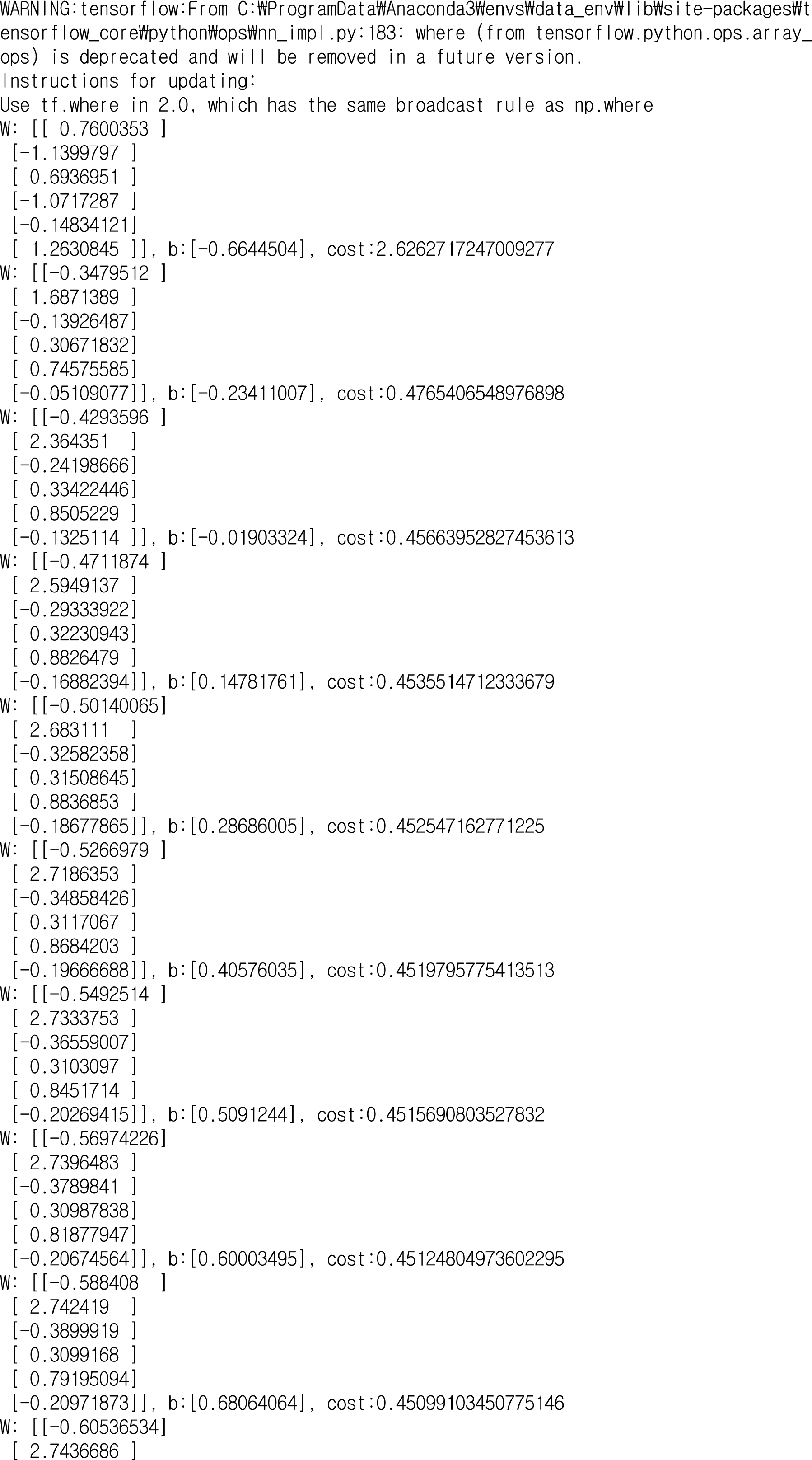


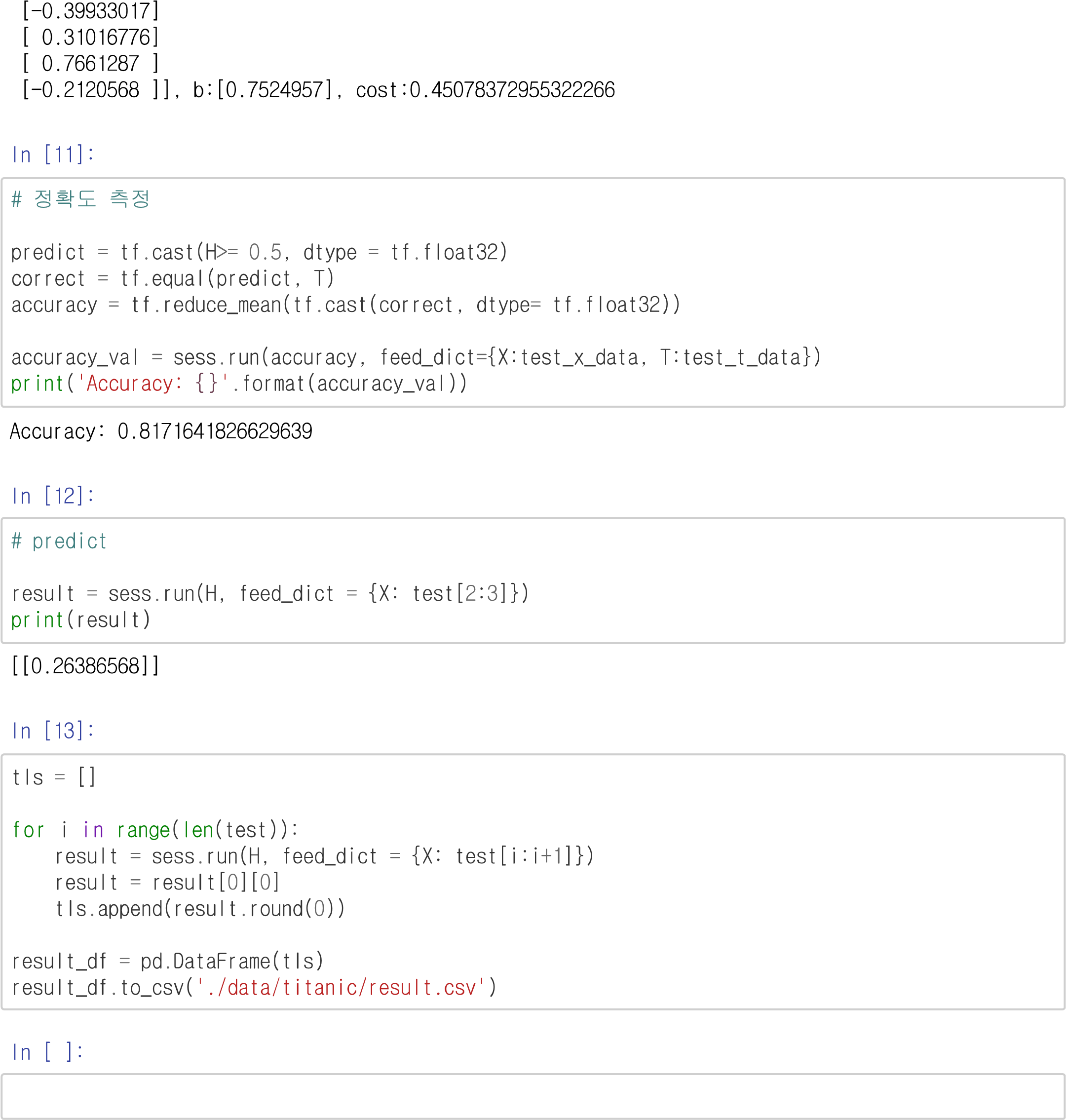












**05. ML EXCERCISE03\_MNIST**

import numpy as np

import pandas as pd

import tensorflow as tf

from sklearn.model\_selection import train\_test\_split

from sklearn.model\_selection import KFold # cross validataion

train = pd.read\_csv('/content/drive/My Drive/Multi Cam/Machine Learning/train.csv')

train.head(2)

# data split

features = list(train)

x\_data\_train, x\_data\_test, t\_data\_train, t\_data\_test = \

train\_test\_split(train[features], train['label'], test\_size = 0.3, random\_state= 0)

# 7:3 비율로 train과 test 데이터를 분리

# onehot\_num

sess = tf.Session()

t\_data\_train\_onehot = sess.run(tf.one\_hot(t\_data\_train, depth = 10)) # depth == 원핫으로 인코딩할 feature 수

t\_data\_test\_onehot = sess.run(tf.one\_hot(t\_data\_test, depth = 10))

# Placeholder

X = tf.placeholder(shape=[None,785], dtype=tf.float32)

T = tf.placeholder(shape=[None, 10], dtype=tf.float32)

# Weight & bias

W = tf.Variable(tf.random.normal([785,10]), name = 'weight')

b = tf.Variable(tf.random.normal([10]), name = 'bias')# 로지스틱 한개에 bias 1개개

# hypothesis

logit = tf.matmul(X,W) + b

H = tf.nn.softmax(logit)

# loss function

loss = tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits\_v2(logits=logit,

                                                                labels = T))

# train node

train = tf.train.GradientDescentOptimizer(learning\_rate=0.01).minimize(loss)

num\_of\_epoch = 9000

batch\_size = 1000 # 한번에 학습할 x\_data와 t\_data의 행의 수

def run\_train(sess, train\_x, train\_t):

    print('### 학습이 시작됨 ###')

    sess.run(tf.global\_variables\_initializer()) # session과 초기화 # 학습 시작전 호출

    total\_batch = int(train\_x.shape[0] / batch\_size)

    for step in range(num\_of\_epoch):

        for i in range(total\_batch):

            batch\_x = train\_x[i\*batch\_size:(i+1)\*batch\_size]

            batch\_t = train\_t[i\*batch\_size:(i+1)\*batch\_size]

            \_, loss\_val = sess.run([train, loss],

                                   feed\_dict = {X: batch\_x,

                                                T: batch\_t})

        if step %1000 ==0:

            print('Loss : {}'.format(loss\_val))

    print( '## 학습 종료 ##')

# 최종 test

run\_train(sess, x\_data\_train, t\_data\_train\_onehot)

“학습이 너무 오래 걸려 마저 학습이 안된채로 12시가 넘을 것 같아 일단 올립니다.”