1. 판다스 문제

1-1. Group Lens의 영화데이터

```
import numpy as np
import pandas as pd


 arr1 = pd.read_csv('./data/ratings.csv')
arr2 = pd.read_csv('./data/movies.csv')


movie_name_rated = pd.merge(arr1,arr2, on='movieId', how='inner')
display(movie_name_rated.head())
```

```
# 1. 사용자가 평가한 모든 영화의 전체 평균 평점


mv1 = movie_name_rated
print('모든 영화의 평점 : ', mv1['rating'].mean())
```

```
# 2. 각 사용자별 평균 평점
mv2_dict = {
      '유저 ID' : [],
      '평균 평점' : []
}
mv2_user = []
mv2_rat = []
```

```
for (user,group) in movie_name_rated.groupby(['userId']):
    mv2_dict['유저 ID'].append(user)
    mv2_dict['평균 평점'].append(group['rating'].mean())

mv2 = pd.DataFrame(mv2_dict).set_index('유저 ID')

display(mv2)
```

```
# 3. 각 영화별 평균 평점
mv3_dict = {
    '영화제목' : [],
    '평균 평점' : []
}

for (mvnm,group) in movie_name_rated.groupby(['title']):
#     print(mvnm)
#     print('==============')
#     print(group.head())
#     print('==============')
    mv3_dict['영화제목'].append(mvnm)
    mv3_dict['평균 평점'].append(group['rating'].mean())

mv3 = pd.DataFrame(mv3_dict).set_index('영화제목')

display(mv3)
```

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

p4 = movie_name_rated['rating'].groupby(movie_name_rated['movieId']).mean()

high_rate = p4[p4 == p4.max()]

print(high_rate)

result = pd.merge(high_rate, arr2, left_index = True, right_on = 'movieId', how='inner')
display(result)
--------------------------------------------------------------------------------
# 5. Comedy영화 중 가장 평점이 낮은 영화의 제목

comedy_movie_id = arr2.loc[arr2['genres'].str.contains('Comedy')]['movieId']
comedy_mask = arr1['movieId'].isin(comedy_movie_id)


low_rating = arr1.loc[comedy_mask].groupby('movieId').mean()['rating'].min()
low_rating_mask = arr1.loc[comedy_mask].groupby('movieId').mean()['rating'] == low_rating

low_movie_id = arr1.loc[comedy_mask].groupby('movieId').mean().loc[low_rating_mask].index
low_movie_id = list(low_movie_id)

display(arr2.loc[arr2['movieId'].isin(low_movie_id)].sort_values(by='title'))
--------------------------------------------------------------------------------
from datetime import datetime
```

```
import time

start_stmp = '2015-01-01 0:0:1'
start_timestamp = time.mktime(datetime.strptime(start_stmp, '%Y-%m-%d %H:%M:%S').timetuple())

end_stmp = '2015-12-31 23:59:59'
end_timestamp = time.mktime(datetime.strptime(end_stmp, '%Y-%m-%d %H:%M:%S').timetuple())

date_mask = ((movie_name_rated['timestamp'].values >= start_timestamp) & (movie_name_rated['timestamp'].values <= end_timestamp))

result1 = movie_name_rated[date_mask]

result1.loc[result1['genres'].str.contains('Romance')]['rating'].mean()
```

1-2. mpg data set문제

```
import numpy as np
import pandas as pd

mpg_data = pd.read_csv('./data/mpg.csv')
display(mpg_data)
```

```
# 1. displ(배기량)이 4 이하인 자동차와 5 이상인 자동차 중
# 어떤 자동차의 hwy(고속도로 연비)가 평균적으로 더 높은지 확인하세요.

mask_1 = mpg_data['displ'].values<=4
mask_2 = mpg_data['displ'].values>=5

if (mpg_data[mask_1]['hwy'].mean()) > (mpg_data[mask_2]['hwy'].mean()):
    print('displ이 4 이하인 자동차의 평균 hwy가 더 높다')
elif (mpg_data[mask_1]['hwy'].mean()) == (mpg_data[mask_2]['hwy'].mean()):
    print('평균 hwy가 같다')
else:
    print('displ이 5 이상인 자동차의 평균 hwy가 더 높다')
print(mpg_data[mask_1]['hwy'].mean())
print(mpg_data[mask_2]['hwy'].mean())
---------------------------------------------------------------------------
# 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.

toyota = mpg_data['manufacturer'] == 'toyota'
audi = mpg_data['manufacturer'] == 'audi'

print(mpg_data[toyota]['cty'].mean())
print(mpg_data[audi]['cty'].mean())

if (mpg_data[toyota]['cty'].mean()) > (mpg_data[audi]['cty'].mean()):
```

```
    print('toyota 자동차의 평균 cty가 더 높다')
elif (mpg_data[toyota]['cty'].mean()) == (mpg_data[audi]['cty'].mean()):
    print('평균 cty가 같다')
else:
    print('audi 자동차의 평균 cty가 더 높다')
```

```
# 3. "chevrolet", "ford", "honda" 자동차의 고속도로 연비 평균을 알아보려고 한다.
# 이 회사들의 데이터를 추출한 후 hwy(고속도로 연비) 평균을 구하세요.

chevrolet = mpg_data['manufacturer'] == 'chevrolet'
ford = mpg_data['manufacturer'] == 'ford'
honda= mpg_data['manufacturer'] == 'honda'

mask_3 = (mpg_data['manufacturer'] == 'chevrolet') | (mpg_data['manufacturer'] == 'ford') | (mpg_data['manufacturer'] == 'honda')

print(' 3개 회사의 자동차의 고속도로 연비 평균 : ', mpg_data[mask_3]['hwy'].mean())
```

```
# 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.

audi_data = mpg_data[audi]

display(audi_data)
```

```
# print(audi_data['hwy'].groupby(audi_data['model']).mean())

display(audi_data.sort_values(by = 'hwy',ascending=False).head())


--------------------------------------------------------------------------------

# 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.


avg_data = (mpg_data['cty'] + mpg_data['hwy'])/2


mpg_data['avg'] = avg_data


suv_id = mpg_data.loc[mpg_data['class'].str.contains('suv')]

display(suv_id.head())


mpg_5 = suv_id['avg'].groupby(suv_id['manufacturer']).mean()

print(mpg_5.sort_values(ascending=False).head())

--------------------------------------------------------------------------------

# 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.


mpg_6 = mpg_data['cty'].groupby(mpg_data['class']).mean()
```

```
print(mpg_6.sort_values(ascending=False))
```

```
# 7. 어떤 회사 자동차의 hwy(고속도로 연비)가 가장 높은지 알아보려 합니다.
# hwy(고속도로 연비) 평균이 가장 높은 회사 세 곳을 출력하세요.

mpg_7 = mpg_data['hwy'].groupby(mpg_data['manufacturer']).mean()
rank_7 = mpg_7.sort_values(ascending=False)

rank_7.index[0]

for x in range(3):
    print([rank_7.index[x] , np.float16(rank_7.values[x]) ])
```

```
# 8. 어떤 회사에서 "compact" 차종을 가장 많이 생산하는지 알아보려고 합니다.
# 각 회사별 "compact" 차종 수를 내림차순으로 정렬해 출력하세요.

compact_id = mpg_data.loc[mpg_data['class']==('compact')]
display(compact_id.head())

mpg_8 = compact_id.groupby(mpg_data['manufacturer']).size()
print(mpg_8.sort_values(ascending=False))
```

1-3. 한국복지패널데이터

```
import pandas as pd
```

```
import numpy as np
import savReaderWriter

with savReaderWriter.SavReader('./data/Koweps_hpc10_2015_beta1.sav', ioUtf8 = True) as reader:
    df = pd.DataFrame(reader.all(), columns = [s for s in reader.header])

print(df.shape)          # # (16664, 957)
display(df.head())
display(df['h10_g3'].head())   # 성별
display(df['h10_g4'].head())   # 태어난 연도
--------------------------------------------------------------------------------
# 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

df = df.loc[df['p1002_8aq1'] != np.nan]

display(df.head())
print(df['p1002_8aq1'].groupby(df['h10_g3']).mean())
--------------------------------------------------------------------------------
# 2. 나이와 월급의 관계
# 몇 살 때 월급을 가장 많이 받을까? 또 그때의 월급은 얼마인가?

money = df.groupby(df['h10_g4']).mean().sort_values(by='p1002_8aq1',ascending=False)
```

```
age = 2020 - (money['p1002_8aq1'].index[0]) +1
print('{}살 때 약 {}만 원을 번다.'.format(int(age), int(money['p1002_8aq1'].values[0])))
--------------------------------------------------------------------------------
# 3. 연령대에 따른 월급 차이
# 30세 미만을 초년(young),     1992 ~
# 30~59세 : 중년(middle),       1962~1991
# 60세 이상 : 노년(old)       ~1961
# 위의 범주로 연령대에 따른 월급의 차이를 알아보자

young = df.loc[(df['h10_g4']>=1992)]
middle = df.loc[(df['h10_g4']<=1991)&(df['h10_g4']>=1962)]
old = df.loc[(df['h10_g4']<=1961)]

print('================================================')
print('초년의 월급 : ',young['p1002_8aq1'].mean())
print('================================================')
print('중년의 월급 : ',middle['p1002_8aq1'].mean())
print('================================================')
print('노년의 월급 : ',old['p1002_8aq1'].mean())
print('================================================')
--------------------------------------------------------------------------------
# 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,..)

```
young_man = young.loc[(young['h10_g3']==1)]
young_woman = young.loc[(young['h10_g3']==2)]

middle_man = middle.loc[(middle['h10_g3']==1)]
middle_woman = middle.loc[(middle['h10_g3']==2)]

old_man = old.loc[(old['h10_g3']==1)]
old_woman = old.loc[(old['h10_g3']==2)]



print('================================================')
print('초년 남성의 월급 {}만원, 여성의 월급 {}만원 '.format(np.float16(young_man['p1002_8aq1'].mean()),np.float16(young_woman['p1002_8aq1'].mean())))
print('================================================')
print('중년 남성의 월급 {}만원, 여성의 월급 {}만원 '.format(np.float16(middle_man['p1002_8aq1'].mean()),np.float16(middle_woman['p1002_8aq1'].mean())))
print('================================================')
print('노년 남성의 월급 {}만원, 여성의 월급 {}만원 '.format(np.float16(old_man['p1002_8aq1'].mean()),np.float16(old_woman['p1002_8aq1'].mean())))
print('================================================')
```

---

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.

df1 = pd.read_excel("./data/Koweps_Codebook.xlsx", sheet_name='직종 코드')

job_money = df.groupby(df['h10_eco9'])
best_money = job_money['p1002_8aq1'].mean().sort_values(ascending=False)
display(best_money)

best_money_job = df1.loc[df1['code_job'] == best_money.index[0]]['job'].values
print('월급을 가장 많이 받는 직업군 : ',best_money_job )
------------------------------------------------------------------------
########################################################################
##################################
# 6. 성별 직업 빈도
# 성별로 어떤 직업이 가장 많을까?

# display(job_money.loc[(job_money['h10_g3'] == 1)])

m_df = df.loc[(df['h10_g3'] == 1)]
w_df = df.loc[(df['h10_g3'] == 2)]

m_df_d = m_df['h10_eco9'].dropna()
display(m_df_d)
w_df_d = w_df['h10_eco9'].dropna()
```

```
display(w_df_d)


man_p6 ={}
woman_p6 ={}


for k in m_df_d:
    if k not in man_p6:
        man_p6[k] = 1
    elif k in man_p6:
        man_p6[k] += 1


s1 = sorted(man_p6.items(), reverse = True ,key = lambda x : x[1])
best_money_m = df1.loc[df1['code_job'] == s1[0][0]]['job'].values
print(s1)


for j in w_df_d:
    if j not in woman_p6:
        woman_p6[j] = 1
    elif j in woman_p6:
        woman_p6[j] += 1



s2 = sorted(woman_p6.items(), reverse = True ,key = lambda x : x[1])
best_money_w = df1.loc[df1['code_job'] == s2[0][0]]['job'].values
print(s2)
```

```
print(best_money_m)

print(best_money_w)


--------------------------------------------------------------------------------

# 7. 종교 유무에 따른 이혼율

# 종교가 있는 사람들이 이혼을 덜 할까??


rel_Y = df.loc[(df['h10_g11'] == 1)]

rel_N = df.loc[(df['h10_g11'] == 2)]


#display(rel_Y.head())

#display(rel_N.head())


#print(rel_Y['h10_g10'] == 3)

#print(rel_N['h10_g10'] == 3)


# h10_g10 혼인상태,


YD = rel_Y.loc[rel_Y['h10_g10'] == 3]

ND = rel_N.loc[rel_N['h10_g10'] == 3]


print('종교가 있으며 이혼 : ', YD.shape[0])

print('종교가 없으며 이혼 : ', ND.shape[0])

--------------------------------------------------------------------------------

# 8. 지역별 연령대 비율

# 노년층이 많은 지역은 어디일까?
```

```
#df['학점'].groupby(df['학과'])


#display(old)


old_region = old.groupby(old['h10_reg7']).size()


r = old_region.sort_values(ascending=False)


print('1. 서울, 2. 수도권(인천/경기), 3. 부산/경남/울산, 4.대구/경북, 5. 대전/충남, 6. 강원/충북, 7.광주/전남/전북/제주도')

print('{}번 지역이 노년층이 가장 많다'.format(r.index[0]))

--------------------------------------------------------------------------------
```

2. titanic data

```
import numpy as np
import tensorflow as tf
import pandas as pd
from scipy import stats
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler

tf.reset_default_graph()
train = pd.read_csv('./data/train.csv')

# df2 = pd.read_csv('./data/test.csv')
# display(df2)

# PassengerId : 승객일련번호 _ 필요 X

# Survived : 생존여부 _ label

# Pclass : 객실 등급
#              상관계수를 구해보고 연관성을 파악해서 살린다.

# Name : 이름 _ 필요 X

# Age : 나이 _ 필요 O
```

```
#          결측치 포함되어 있다.
#          1. 모든 사람의 평균
#          2. 이름에 포함된 MR, MRS, MISS를 고려해 평균나이 사용
#          유아(0) 청소년(1) 성인(2) 노인(3)

# Sex : male, Female => 0,1로 변경

# SibSp, Parch : column을 합쳐 num_of_family

# Ticket : 삭제
# Fare : 요금 _ 고민을 해 봐야 한다.

# Cabin : 객실번호 X

# Enbarked : 승선위치 S,C,Q  => 숫자로 바꿔서 처리

#########################################################################

# 필요없는 column 삭제
train.drop(['PassengerId','Name','Ticket','Fare', 'Cabin'],axis=1,inplace=True)

# 성별 처리
sex_mapping = {'male':0, 'female':1}
train['Sex'] = train['Sex'].map(sex_mapping)
```

```
# 가족 구성원
train['Family'] = train['SibSp']+train['Parch']
train.drop(['SibSp', 'Parch'],axis=1,inplace=True)

# 결측치
# train.isnull().sum()
# Survived          0
# Pclass            0
# Sex               0
# Age             177
# Embarked          2
# Family            0

# Embarked  결측치
train['Embarked'] = train['Embarked'].fillna('Q')

# Embarked 변환
Embarked_mapping = {'S':0, 'C':1, 'Q':2}
train['Embarked'] = train['Embarked'].map(Embarked_mapping)

# Age 결측치
train['Age'] = train['Age'].fillna(train['Age'].mean())

# Age에 대해 Binning 처리 (Numerical value -> Categorical value)
train.loc[train['Age']<8,'Age']=0
train.loc[(train['Age']>=8) & (train['Age'] < 20) ,'Age']=1
```

```
train.loc[(train['Age']>=20) & (train['Age'] < 65) ,'Age']=2

train.loc[train['Age']>=65,'Age']=3

display(train.head())



# 데이터 준비 완료

# Tensorflow를 이용해서 학습 진행 및 Accuracy를 구현



#train = train.drop('Survived',axis=1, inplace = False).values.reshape(-1,5)



brd = int(train.shape[0]*0.7)



train_data = train.iloc[0:brd]

test_data = train.iloc[brd:]



x_data_train = train_data.drop('Survived',axis=1, inplace = False).values.reshape(-1,5)

t_data_train = train_data['Survived'].values.reshape(-1,1)



x_data_test = test_data.drop('Survived',axis=1, inplace = False).values.reshape(-1,5)

t_data_test = test_data['Survived'].values.reshape(-1,1)



# Tensorflow



X = tf.placeholder(shape=[None,5], dtype= tf.float32)
```

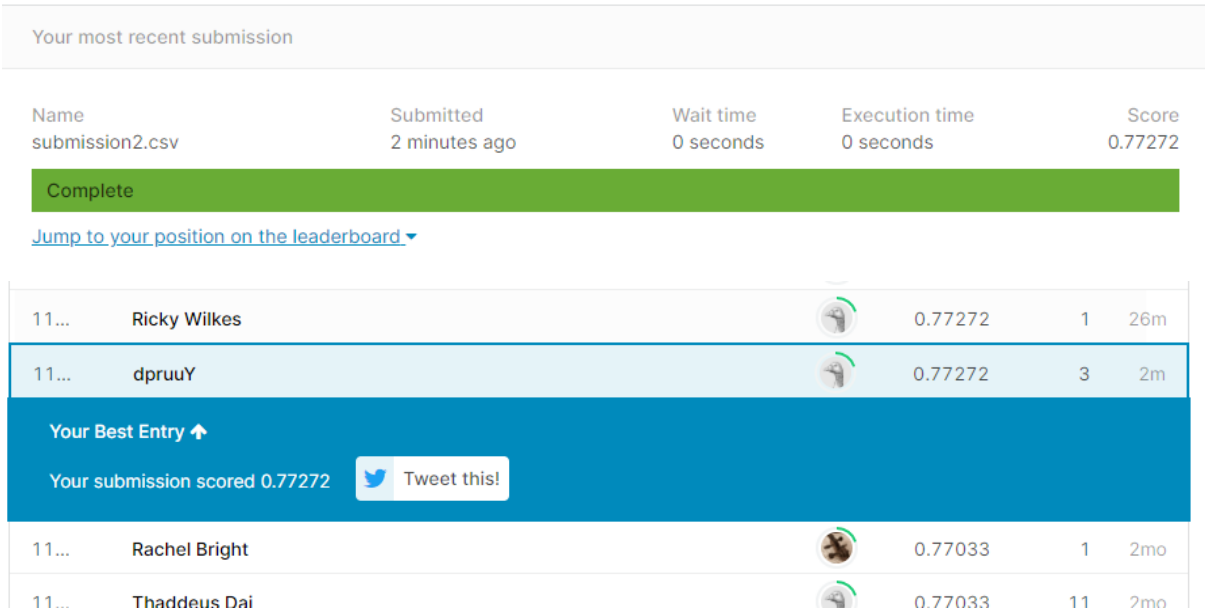
```
T = tf.placeholder(shape=[None,1], dtype= tf.float32)

W = tf.Variable(tf.random.normal([5,1]),name='Weight')
b = tf.Variable(tf.random.normal([1]),name='bias')

# # Hypothesis
# logit = tf.matmul(X,W)+b
# H = tf.sigmoid(logit)

drop_rate = tf.placeholder(dtype = tf.float32)

# Weight & bias

W2 = tf.get_variable('weight2', shape=[5,5],
                     initializer = tf.contrib.layers.variance_scaling_initializer())
b2 = tf.Variable(tf.random.normal([5]), name='bias2')

_layer2 = tf.nn.relu(tf.matmul(X,W2) + b2)
layer2 = tf.nn.dropout(_layer2, rate = drop_rate)

W3 = tf.get_variable('weight3', shape = [5,5],
                     initializer = tf.contrib.layers.variance_scaling_initializer())
b3 = tf.Variable(tf.random.normal([5]), name='bias3')

_layer3 = tf.nn.relu(tf.matmul(layer2,W3) + b3)
layer3 = tf.nn.dropout(_layer3, rate = drop_rate)
```

```
W4 = tf.get_variable('weight4', shape = [5,1],
                                initializer = tf.contrib.layers.variance_scaling_initializer())
b4 = tf.Variable(tf.random.normal([1]), name='bias4')



# Hypothesis
logit = tf.matmul(layer3,W4) + b4
H = tf.nn.relu(logit)     # softmax activation function



# loss function
loss = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(logits=logit,labels=T))

# train
train = tf.train.GradientDescentOptimizer(learning_rate=1e-1).minimize(loss)

# session
sess = tf.Session()
sess.run(tf.global_variables_initializer())

# 학습
for step in range(300000):
        _,W_val,b_val,loss_val = sess.run([train,W,b,loss],feed_dict={X:x_data_train,
                                                                      T:t_data_train,
                                                                     drop_rate : 0.1})
```

```
            if step % 30000 ==0:
                print('W : {}, b : {}, loss : {}'.format(W_val,b_val,loss_val))


predict = tf.cast(H >= 0.5, dtype=tf.float32)  # True = 1, False = 0
correct = tf.equal(predict,T)
accuracy = tf.reduce_mean(tf.cast(correct, dtype=tf.float32))

accuracy_val = sess.run(accuracy, feed_dict={X:x_data_test,T:t_data_test, drop_rate : 0})
print('Accuracy : {}'.format(accuracy_val))

test = pd.read_csv('./data/test.csv')

PassengerId = test['PassengerId']
test.drop(['PassengerId','Name','Ticket','Fare', 'Cabin'],axis=1,inplace=True)

# 성별 처리
sex_mapping = {'male':0, 'female':1}
test['Sex'] = test['Sex'].map(sex_mapping)

# 가족 구성원
test['Family'] = test['SibSp']+test['Parch']
test.drop(['SibSp', 'Parch'],axis=1,inplace=True)

# Embarked  결측치
```

```
test['Embarked'] = test['Embarked'].fillna('Q')

# Embarked 변환
Embarked_mapping = {'S':0, 'C':1, 'Q':2}
test['Embarked'] = test['Embarked'].map(Embarked_mapping)

# Age 결측치
test.loc[test['Age'].isnull(),'Age'] = np.nanmedian(test['Age'].values)

# Age에 대해 Binning 처리 (Numerical value -> Categorical value)
test.loc[test['Age']<8,'Age']=0
test.loc[(test['Age']>=8) & (test['Age'] < 20) ,'Age']=1
test.loc[(test['Age']>=20) & (test['Age'] < 65) ,'Age']=2
test.loc[test['Age']>=65,'Age']=3

test_x = test.values.reshape(-1,5)


predict = tf.cast(H >= 0.5, dtype=tf.float32)  # True = 1, False = 0

results = sess.run(predict, feed_dict={X:test_x,drop_rate : 0})
# results = model.predict(test)

# select the indix with the maximum probability
```

```
#results = pd.Series(results.ravel(),name="Survived")

#submission = pd.concat([pd.Series(range(892,1310),name = "PassengerId"),results],axis = 1)

submission = pd.DataFrame(
    {
        'PassengerId': PassengerId,
        'Survived': results.ravel().astype(int)
    }
)

submission.to_csv("submission2.csv",index=False)
```



3. mnist data

```
import numpy as np

import pandas as pd

import tensorflow as tf

from scipy import stats


from sklearn.model_selection import train_test_split

from sklearn.preprocessing import MinMaxScaler

from sklearn.model_selection import KFold


train = pd.read_csv('./data/digit-recognizer/train.csv')

test =   pd.read_csv('./data/digit-recognizer/test.csv')

display(train.head())

display(test.head())


# 1. 결측치 확인

train.isnull().sum()


# zscore = 1.8

# for t in train.columns:

#         df.loc[np.abs(stats.zscore(df[t])) >= zscore,:]



train_x = train.drop('label',axis=1, inplace = False)
```

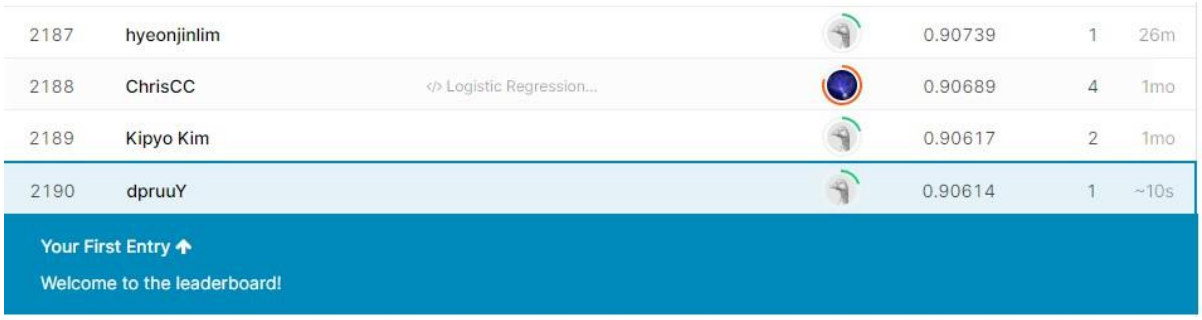
```
train_t = train['label']

x_data_train, x_data_test, t_data_train, t_data_test = ₩
train_test_split(train_x,train_t,test_size=0.3, random_state=0)

scaler = MinMaxScaler()
scaler.fit(x_data_train)

x_data_train_norm = scaler.transform(x_data_train)
x_data_test_norm = scaler.transform(x_data_test)

sess = tf.Session()

t_data_train_onehot = sess.run(tf.one_hot(t_data_train, depth=10))
t_data_test_onehot = sess.run(tf.one_hot(t_data_test, depth=10))

del t_data_train
del t_data_test

# Training data set 준비완료

# Tensorflow Graph 그리기

# Placeholder

X = tf.placeholder(shape=[None,784],dtype=tf.float32)
```

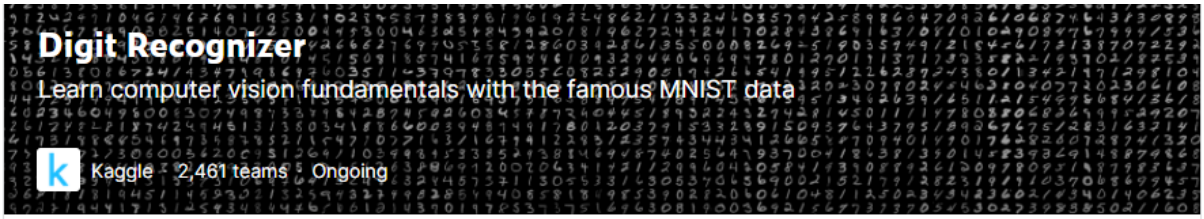
```
T = tf.placeholder(shape=[None,10],dtype=tf.float32)


# Weight & bias


W = tf.Variable(tf.random.normal([784,10]), name = 'weight')

b = tf.Variable(tf.random.normal([10]), name = 'bias')


# Hypothesis(Model)

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

train = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(loss)



# 반복학습하는 함수


num_of_epoch = 1000

batch_size = 100 # 한번에 학습할 x_data, t_data의 행의 수 16000개중 100개 씩 학습


def run_train(sess, train_x, train_t):

    print('### 학습 시작 ###')

    sess.run(tf.global_variables_initializer()) # session 초기화
```

```
    total_batch = int(train_x.shape[0] / batch_size)

    for step in range(num_of_epoch):

        for i in range(total_batch): # 0 ~ 139
            batch_x = train_x[i*batch_size:(i+1)*batch_size] # train_x[0:100]
            batch_t = train_t[i*batch_size:(i+1)*batch_size] # train_t[0:100]
            _, loss_val = sess.run([train,loss], feed_dict={X:batch_x ,
                                                            T:batch_t})

        if step % 100 == 0:
            print('Loss : {}'.format(loss_val))

    print('### 학습 종료 ###')



# Accuracy (정확도 측정)      #      0      1      2
predict = tf.argmax(H,1)      # [[0.5   0.4   0.1]]  입력 값에 대한 예측
correct = tf.equal(predict, tf.argmax(T,1))
accuracy = tf.reduce_mean(tf.cast(correct,dtype=tf.float32))



# # training data로 validation
# run_train(sess,x_data_train_norm,t_data_train_onehot)
# # Accuracy 측정 (training data로 validation)
```

```
# result = sess.run(accuracy, feed_dict={X:x_data_train_norm    ,
#                                        T:t_data_train_onehot})

# print('training data로 validation {}'.format(result))
# # 비추천
# # Cross Validation 추천


# Cross Validation
cv= 5
results = []
kf = KFold(n_splits=cv, shuffle=True)

for training_idx, validation_idx in kf.split(x_data_train_norm):
    # training_idx : index 값을 알아온다
    train_x = x_data_train_norm[training_idx]
    train_t = t_data_train_onehot[training_idx]

    valid_x = x_data_train_norm[validation_idx]
    valid_t = t_data_train_onehot[validation_idx]

    run_train(sess,train_x,train_t)
    results.append(sess.run(accuracy,
                            feed_dict={X:valid_x,
                                       T:valid_t}))
```

```
print('Cross Validation 결과 : {}'.format(results))

print('Cross Validation 최종 결과 : {}'.format(np.mean(results)))


result = sess.run(accuracy, feed_dict={X:x_data_test_norm    ,
                                       T:t_data_test_onehot})


print('최종 정확도 {}'.format(result))


# predict results

my_state_scaled = scaler.transform(test)

results = sess.run(H, feed_dict={X:my_state_scaled})

# results = model.predict(test)


# select the indix with the maximum probability

results = np.argmax(results,axis = 1)


results = pd.Series(results,name="Label")

submission = pd.concat([pd.Series(range(1,28001),name = "ImageId"),results],axis = 1)


submission.to_csv("result.csv",index=False)
```





1 submissions for dpruuY

Sort by Most recent

All Successful Selected

Submission and Description	Public Score
result.csv 8 days ago by dpruuY practice	0.90614

No more submissions to show