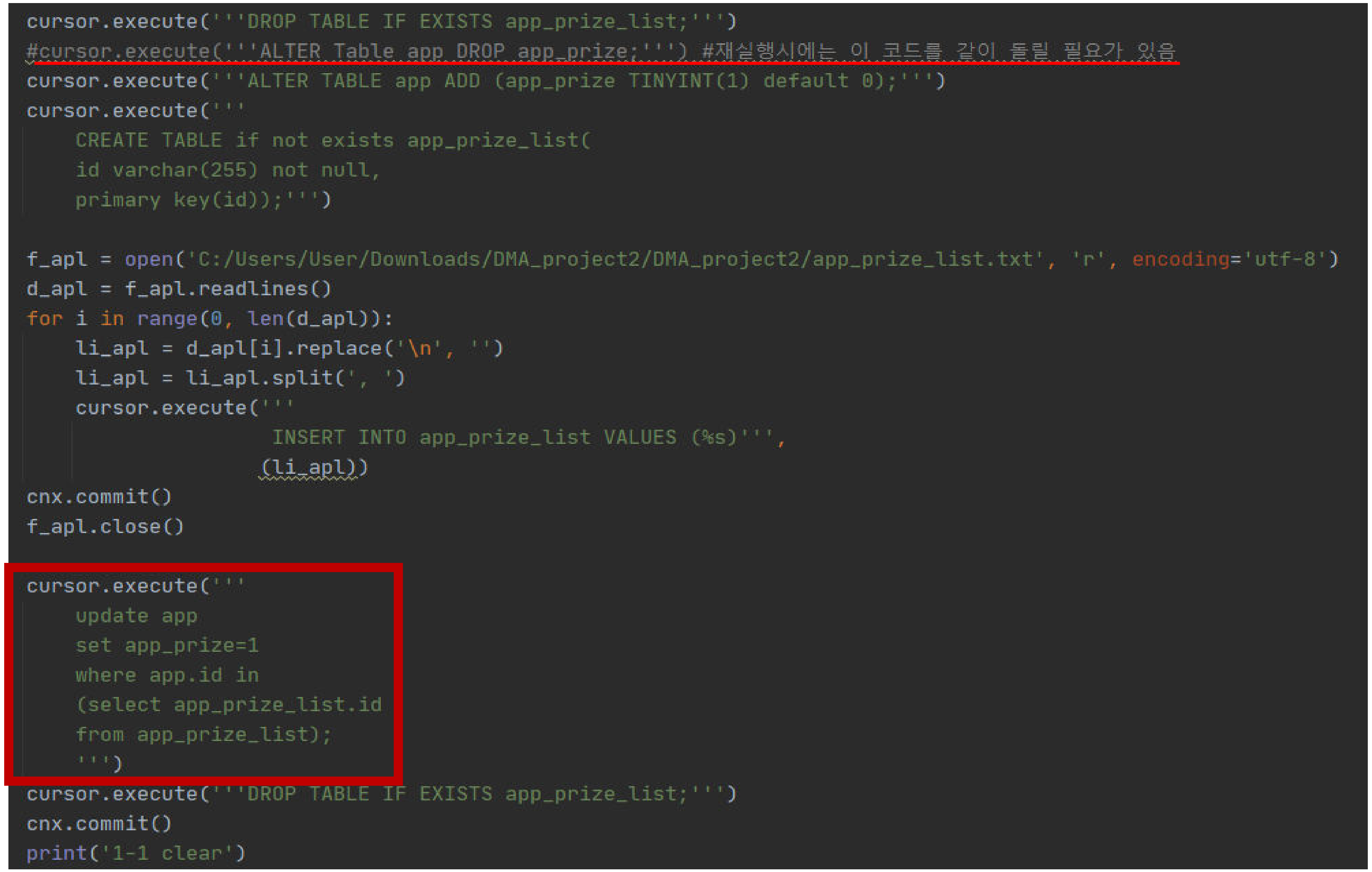
데이터관리와 분석

프로젝트#2: DB mining & Automated Recommendation System

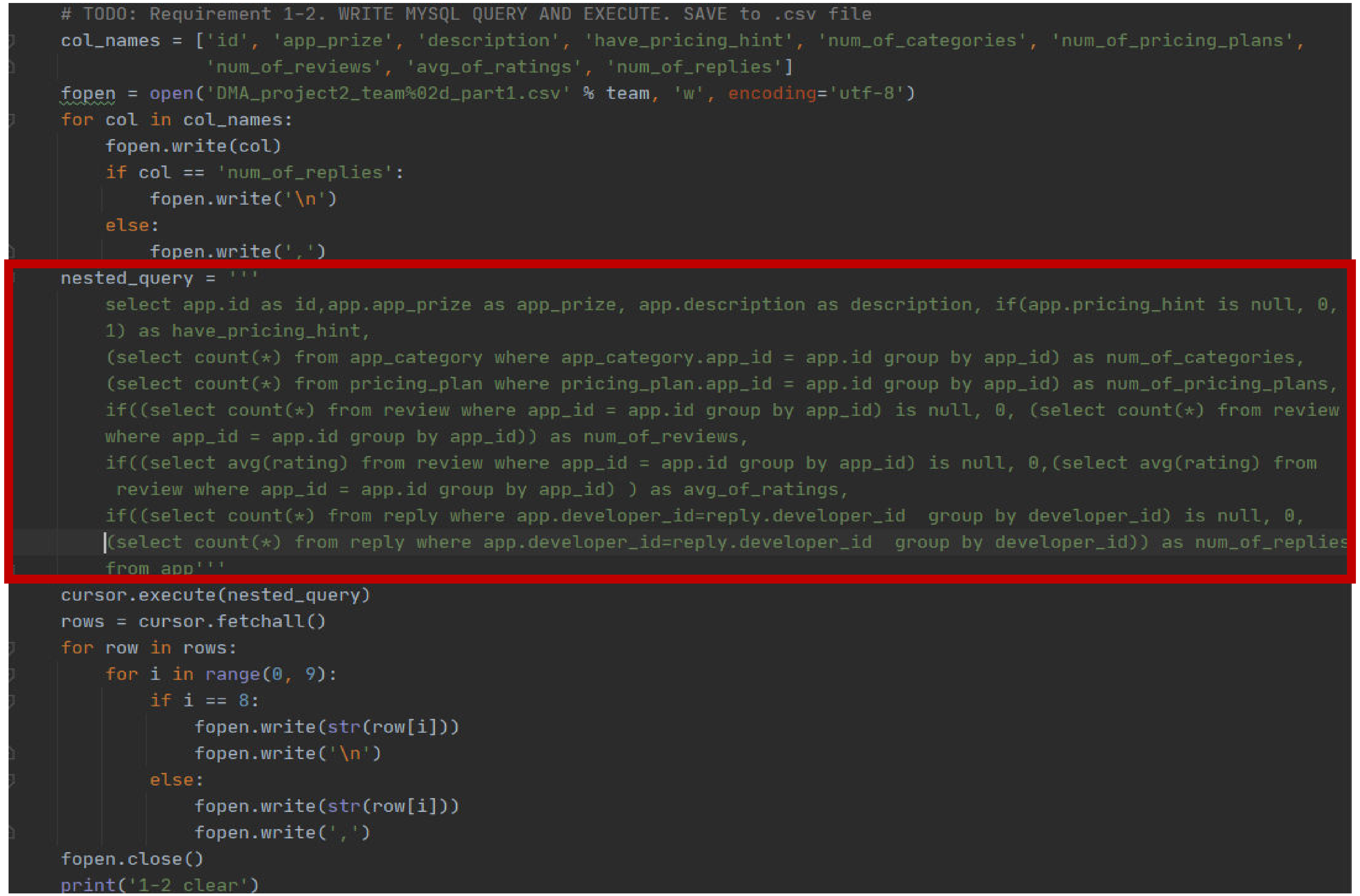
12조
2017-15751 장준혁
2017-18868 조형찬
2017-10854 최의현
2017-13918 황재훈
2017-10089 윤성진

R1-1



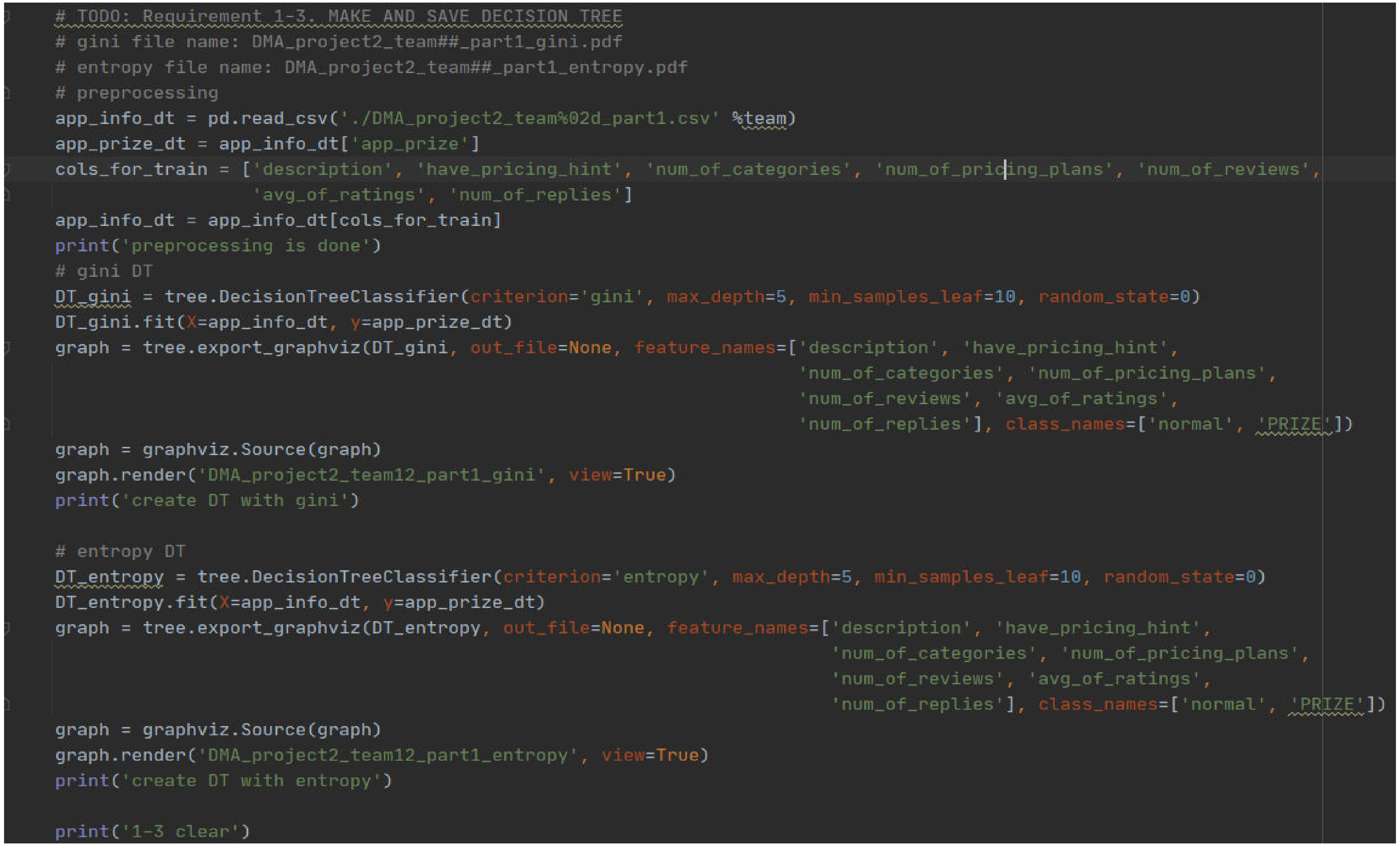
```python
cursor.execute('''DROP TABLE IF EXISTS app_prize_list;''')
#cursor.execute('''ALTER Table app DROP app_prize;''') #재실행시에는 이 코드를 같이 돌릴 필요가 있음
cursor.execute('''ALTER TABLE app ADD (app_prize TINYINT(1) default 0);''')
cursor.execute('''
    CREATE TABLE if not exists app_prize_list(
    id varchar(255) not null,
    primary key(id));''')

f_apl = open('C:/Users/User/Downloads/DMA_project2/DMA_project2/app_prize_list.txt', 'r', encoding='utf-8')
d_apl = f_apl.readlines()
for i in range(0, len(d_apl)):
    li_apl = d_apl[i].replace('\n', '')
    li_apl = li_apl.split(', ')
    cursor.execute('''
                    INSERT INTO app_prize_list VALUES (%s)''',
                   (li_apl))
cnx.commit()
f_apl.close()

cursor.execute('''
    update app
    set app_prize=1
    where app.id in
    (select app_prize_list.id
    from app_prize_list);
    ''')
cursor.execute('''DROP TABLE IF EXISTS app_prize_list;''')
cnx.commit()
print('1-1 clear')
```

R1-2



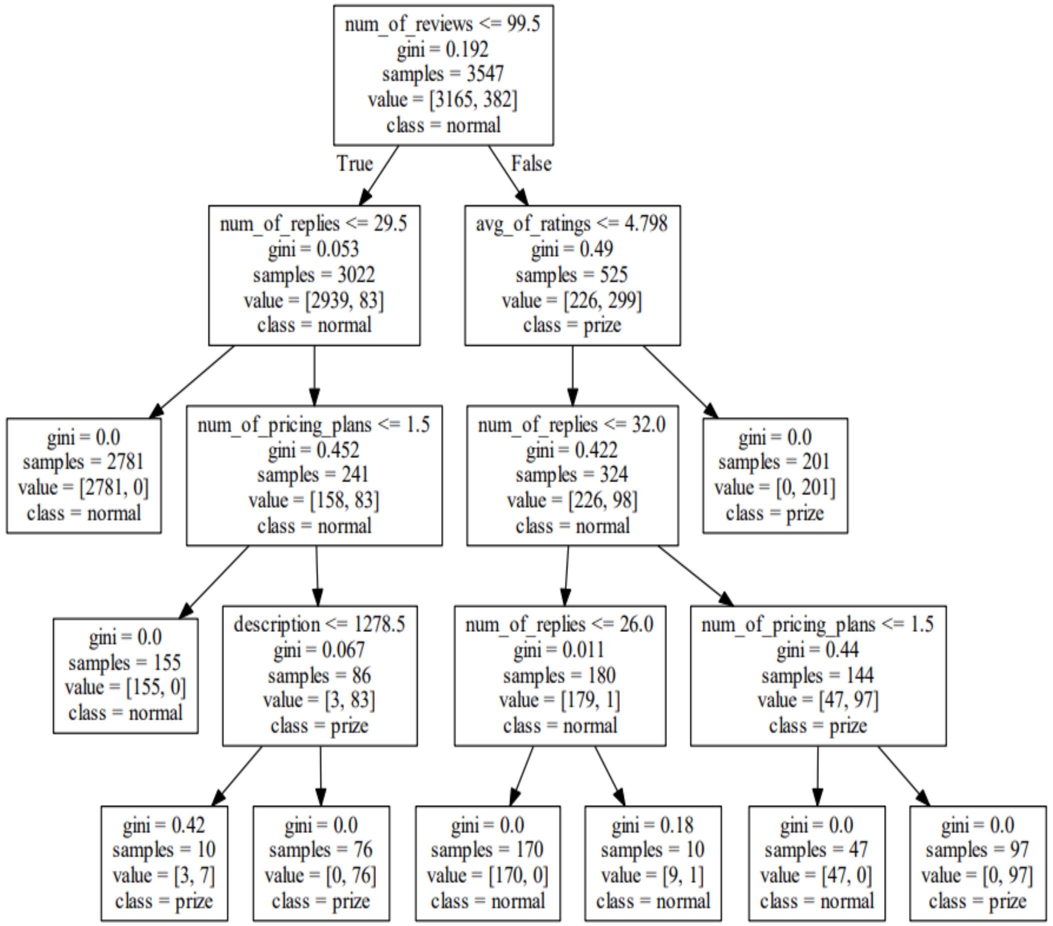
```python
# TODO: Requirement 1-2. WRITE MYSQL QUERY AND EXECUTE. SAVE to .csv file
col_names = ['id', 'app_prize', 'description', 'have_pricing_hint', 'num_of_categories', 'num_of_pricing_plans',
             'num_of_reviews', 'avg_of_ratings', 'num_of_replies']
fopen = open('DMA_project2_team%02d_part1.csv' % team, 'w', encoding='utf-8')
for col in col_names:
    fopen.write(col)
    if col == 'num_of_replies':
        fopen.write('\n')
    else:
        fopen.write(',')
nested_query = '''
    select app.id as id,app.app_prize as app_prize, app.description as description, if(app.pricing_hint is null, 0,
    1) as have_pricing_hint,
    (select count(*) from app_category where app_category.app_id = app.id group by app_id) as num_of_categories,
    (select count(*) from pricing_plan where pricing_plan.app_id = app.id group by app_id) as num_of_pricing_plans,
    if((select count(*) from review where app_id = app.id group by app_id) is null, 0, (select count(*) from review
    where app_id = app.id group by app_id)) as num_of_reviews,
    if((select avg(rating) from review where app_id = app.id group by app_id) is null, 0,(select avg(rating) from
     review where app_id = app.id group by app_id) ) as avg_of_ratings,
    if((select count(*) from reply where app.developer_id=reply.developer_id  group by developer_id) is null, 0,
    (select count(*) from reply where app.developer_id=reply.developer_id  group by developer_id)) as num_of_replies
    from app'''
cursor.execute(nested_query)
rows = cursor.fetchall()
for row in rows:
    for i in range(0, 9):
        if i == 8:
            fopen.write(str(row[i]))
            fopen.write('\n')
        else:
            fopen.write(str(row[i]))
            fopen.write(',')
fopen.close()
print('1-2 clear')
```

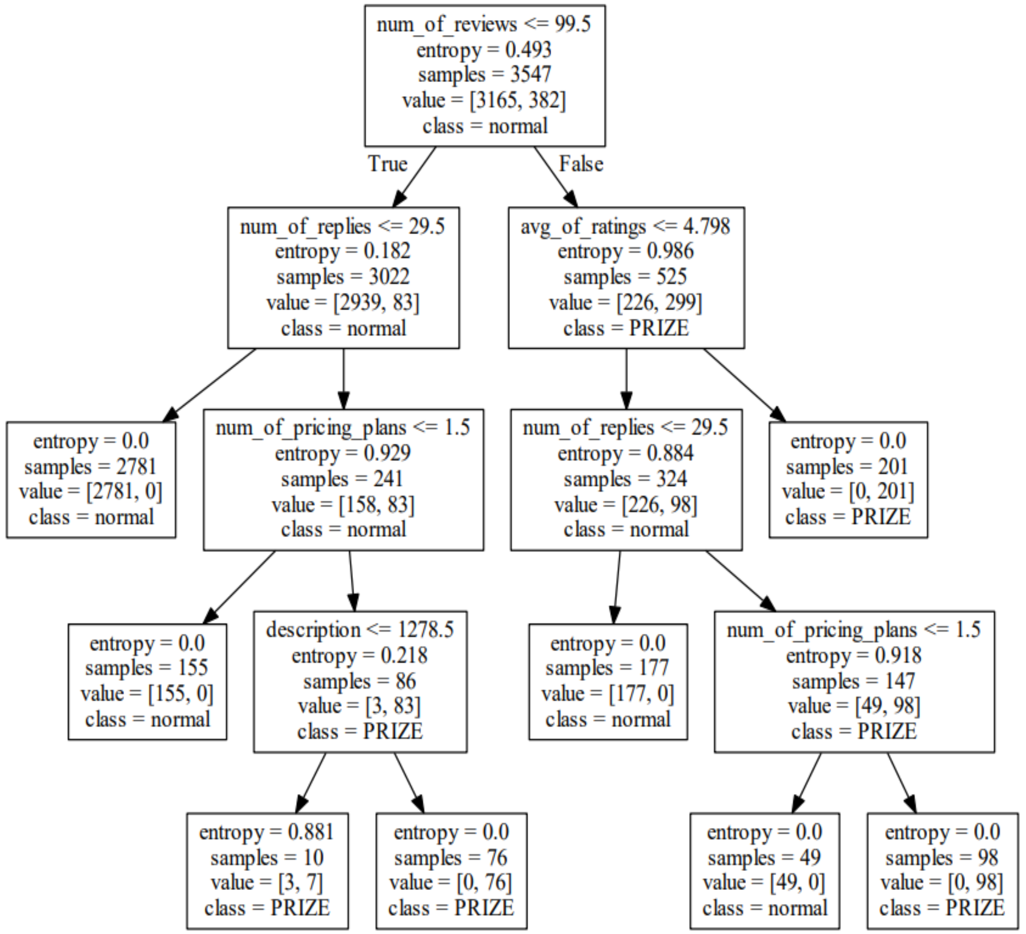
R1-3



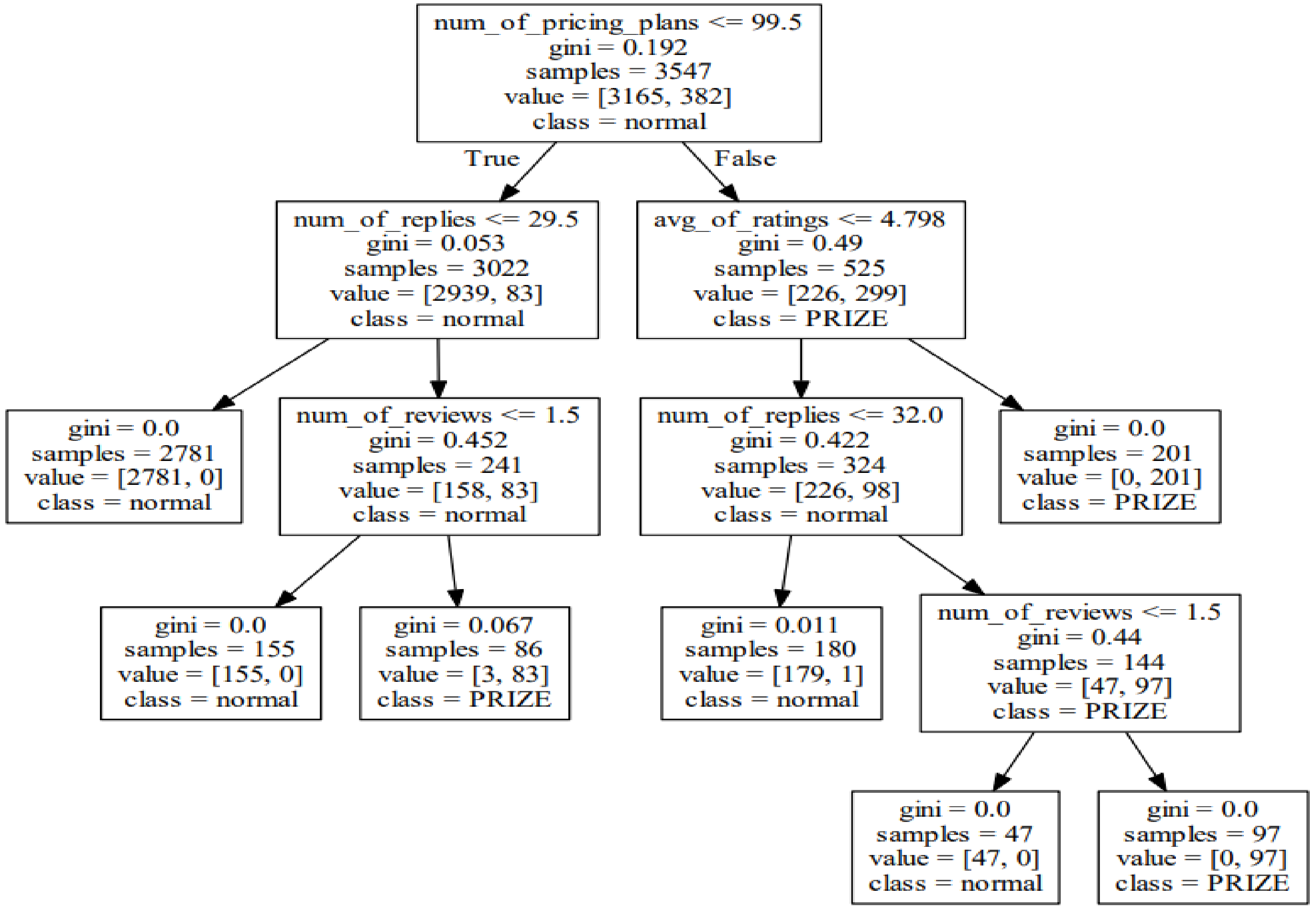
```python
# TODO: Requirement 1-3. MAKE AND SAVE DECISION TREE
# gini file name: DMA_project2_team##_part1_gini.pdf
# entropy file name: DMA_project2_team##_part1_entropy.pdf
# preprocessing
app_info_dt = pd.read_csv('./DMA_project2_team%02d_part1.csv' %team)
app_prize_dt = app_info_dt['app_prize']
cols_for_train = ['description', 'have_pricing_hint', 'num_of_categories', 'num_of_pricing_plans', 'num_of_reviews',
                  'avg_of_ratings', 'num_of_replies']
app_info_dt = app_info_dt[cols_for_train]
print('preprocessing is done')
# gini DT
DT_gini = tree.DecisionTreeClassifier(criterion='gini', max_depth=5, min_samples_leaf=10, random_state=0)
DT_gini.fit(X=app_info_dt, y=app_prize_dt)
graph = tree.export_graphviz(DT_gini, out_file=None, feature_names=['description', 'have_pricing_hint',
                                                                    'num_of_categories', 'num_of_pricing_plans',
                                                                    'num_of_reviews', 'avg_of_ratings',
                                                                    'num_of_replies'], class_names=['normal', 'PRIZE'])
graph = graphviz.Source(graph)
graph.render('DMA_project2_team12_part1_gini', view=True)
print('create DT with gini')

# entropy DT
DT_entropy = tree.DecisionTreeClassifier(criterion='entropy', max_depth=5, min_samples_leaf=10, random_state=0)
DT_entropy.fit(X=app_info_dt, y=app_prize_dt)
graph = tree.export_graphviz(DT_entropy, out_file=None, feature_names=['description', 'have_pricing_hint',
                                                                       'num_of_categories', 'num_of_pricing_plans',
                                                                       'num_of_reviews', 'avg_of_ratings',
                                                                       'num_of_replies'], class_names=['normal', 'PRIZE'])
graph = graphviz.Source(graph)
graph.render('DMA_project2_team12_part1_entropy', view=True)
print('create DT with entropy')

print('1-3 clear')
```

R1-3 결과


num_of_reviews <= 99.5
gini = 0.192
samples = 3547
value = [3165, 382]
class = normal
True
False
num_of_replies <= 29.5
gini = 0.053
samples = 3022
value = [2939, 83]
class = normal
avg_of_ratings <= 4.798
gini = 0.49
samples = 525
value = [226, 299]
class = prize
gini = 0.0
samples = 2781
value = [2781, 0]
class = normal
num_of_pricing_plans <= 1.5
gini = 0.452
samples = 241
value = [158, 83]
class = normal
num_of_replies <= 32.0
gini = 0.422
samples = 324
value = [226, 98]
class = normal
gini = 0.0
samples = 201
value = [0, 201]
class = prize
gini = 0.0
samples = 155
value = [155, 0]
class = normal
description <= 1278.5
gini = 0.067
samples = 86
value = [3, 83]
class = prize
num_of_replies <= 26.0
gini = 0.011
samples = 180
value = [179, 1]
class = normal
num_of_pricing_plans <= 1.5
gini = 0.44
samples = 144
value = [47, 97]
class = prize
gini = 0.42
samples = 10
value = [3, 7]
class = prize
gini = 0.0
samples = 76
value = [0, 76]
class = prize
gini = 0.0
samples = 170
value = [170, 0]
class = normal
gini = 0.18
samples = 10
value = [9, 1]
class = normal
gini = 0.0
samples = 47
value = [47, 0]
class = normal
gini = 0.0
samples = 97
value = [0, 97]
class = prize


num_of_reviews <= 99.5
entropy = 0.493
samples = 3547
value = [3165, 382]
class = normal
True
False
num_of_replies <= 29.5
entropy = 0.182
samples = 3022
value = [2939, 83]
class = normal
avg_of_ratings <= 4.798
entropy = 0.986
samples = 525
value = [226, 299]
class = PRIZE
entropy = 0.0
samples = 2781
value = [2781, 0]
class = normal
num_of_pricing_plans <= 1.5
entropy = 0.929
samples = 241
value = [158, 83]
class = normal
num_of_replies <= 29.5
entropy = 0.884
samples = 324
value = [226, 98]
class = normal
entropy = 0.0
samples = 201
value = [0, 201]
class = PRIZE
entropy = 0.0
samples = 155
value = [155, 0]
class = normal
description <= 1278.5
entropy = 0.218
samples = 86
value = [3, 83]
class = PRIZE
entropy = 0.0
samples = 177
value = [177, 0]
class = normal
num_of_pricing_plans <= 1.5
entropy = 0.918
samples = 147
value = [49, 98]
class = PRIZE
entropy = 0.881
samples = 10
value = [3, 7]
class = PRIZE
entropy = 0.0
samples = 76
value = [0, 76]
class = PRIZE
entropy = 0.0
samples = 49
value = [49, 0]
class = normal
entropy = 0.0
samples = 98
value = [0, 98]
class = PRIZE

R1-4



#gini모델- 개선 여지 보여 채택

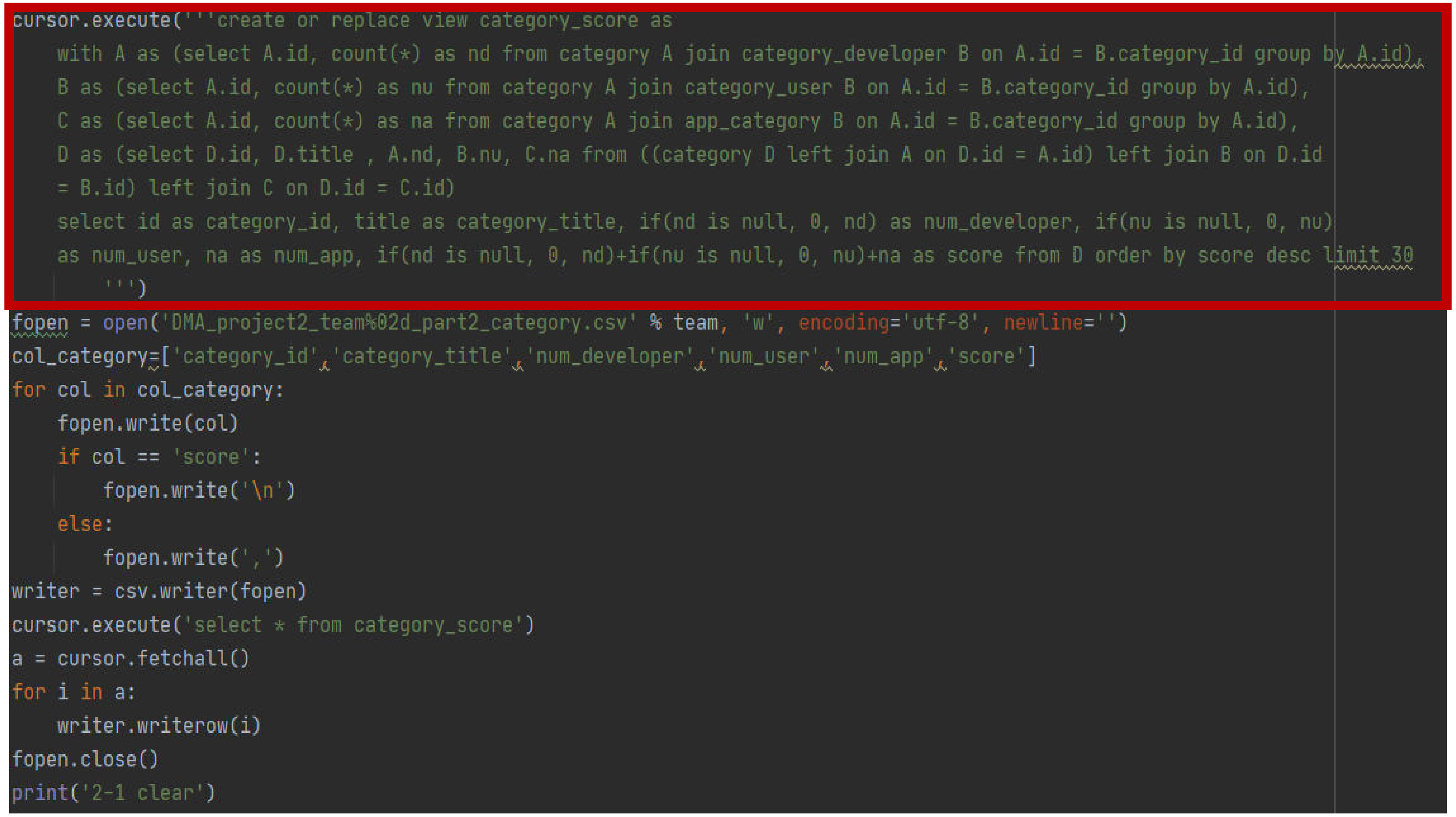
#max-depth=4로 채택(over-fitting 방지)

#Min-samples-leaf 5로 바꿈(상세하게 분류)

#Min-impurity_decrease 0.002로 (overfitting 방지)

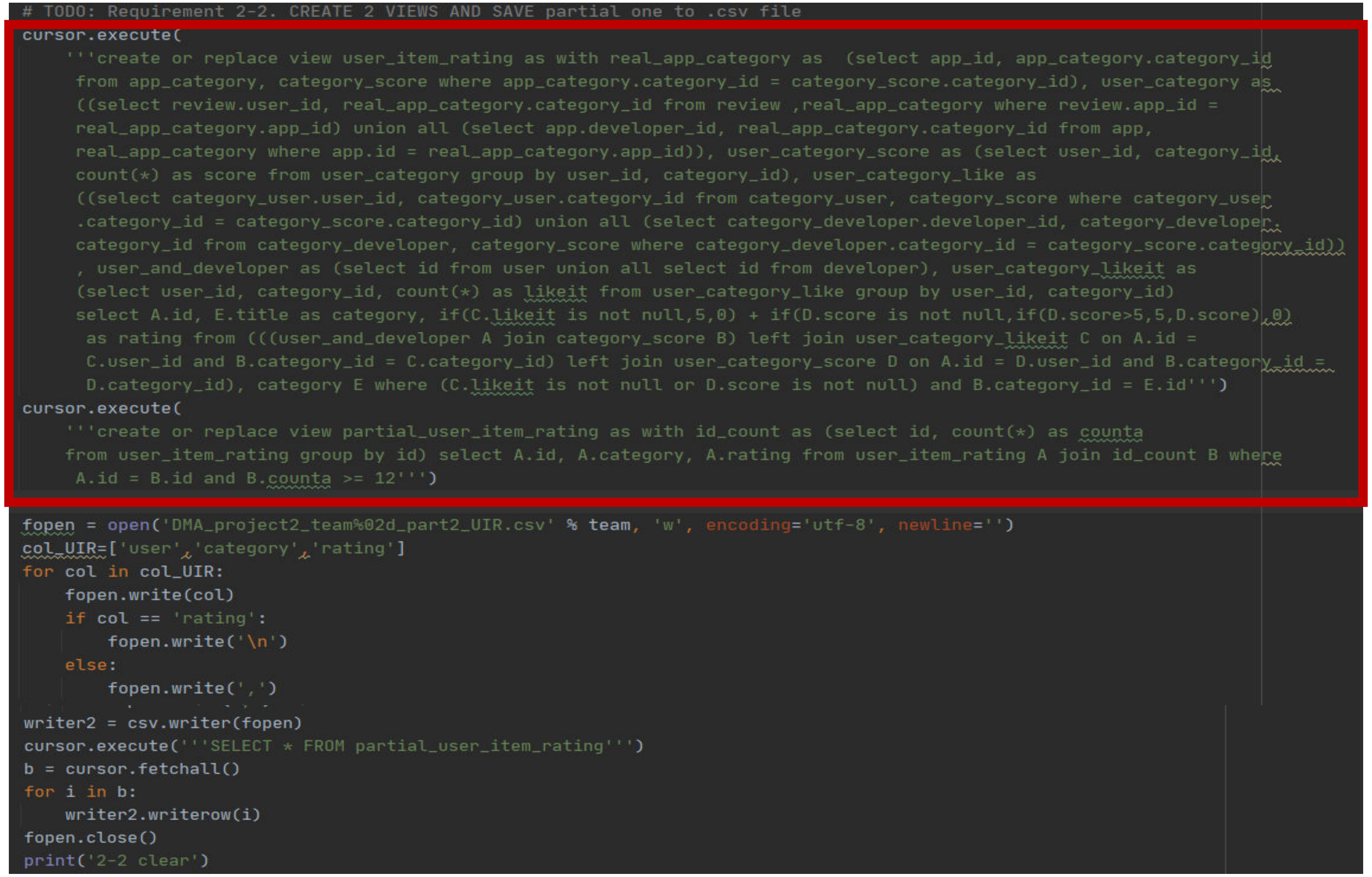
#have_pricing_hint(중요한 기준이 아니라 삭제)

R2-1



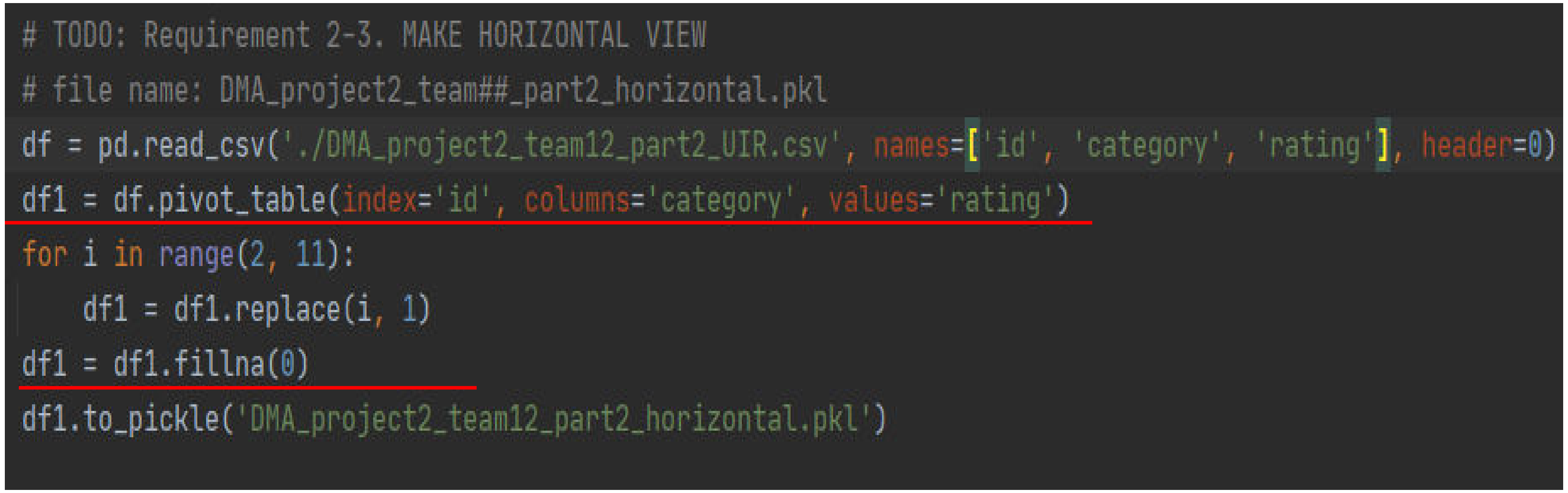
```
cursor.execute('''create or replace view category_score as
    with A as (select A.id, count(*) as nd from category A join category_developer B on A.id = B.category_id group by A.id),
    B as (select A.id, count(*) as nu from category A join category_user B on A.id = B.category_id group by A.id),
    C as (select A.id, count(*) as na from category A join app_category B on A.id = B.category_id group by A.id),
    D as (select D.id, D.title , A.nd, B.nu, C.na from ((category D left join A on D.id = A.id) left join B on D.id
    = B.id) left join C on D.id = C.id)
    select id as category_id, title as category_title, if(nd is null, 0, nd) as num_developer, if(nu is null, 0, nu)
    as num_user, na as num_app, if(nd is null, 0, nd)+if(nu is null, 0, nu)+na as score from D order by score desc limit 30
        ''')
fopen = open('DMA_project2_team%02d_part2_category.csv' % team, 'w', encoding='utf-8', newline='')
col_category=['category_id','category_title','num_developer','num_user','num_app','score']
for col in col_category:
    fopen.write(col)
    if col == 'score':
        fopen.write('\n')
    else:
        fopen.write(',')
writer = csv.writer(fopen)
cursor.execute('select * from category_score')
a = cursor.fetchall()
for i in a:
    writer.writerow(i)
fopen.close()
print('2-1 clear')
```

R2-2



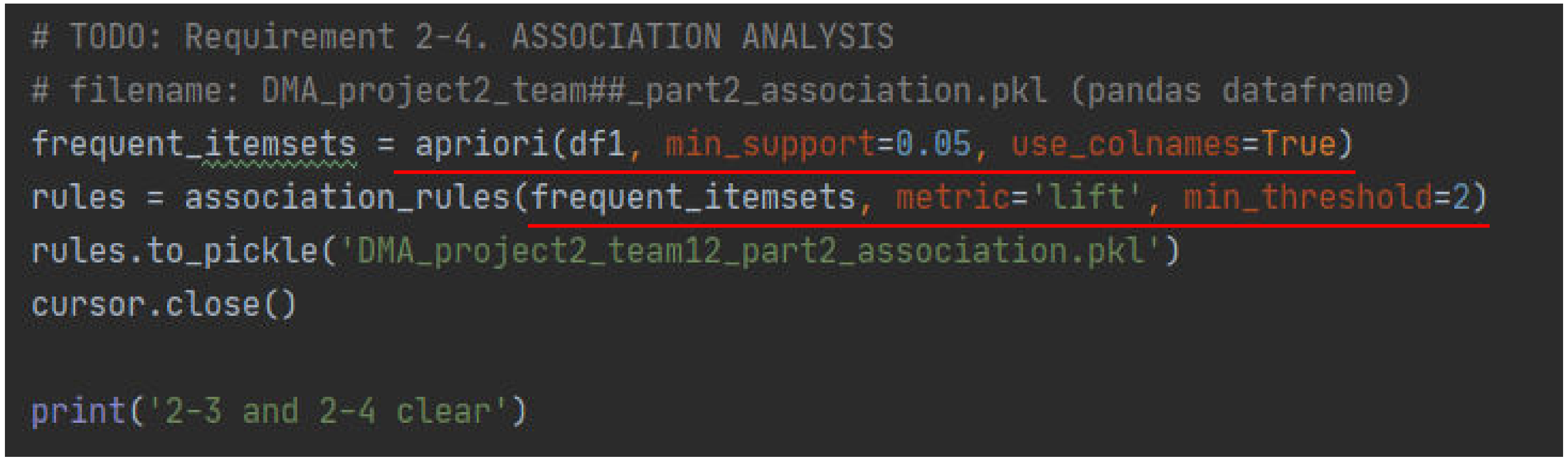
```
# TODO: Requirement 2-2. CREATE 2 VIEWS AND SAVE partial one to .csv file
cursor.execute(
    '''create or replace view user_item_rating as with real_app_category as  (select app_id, app_category.category_id
     from app_category, category_score where app_category.category_id = category_score.category_id), user_category as
     ((select review.user_id, real_app_category.category_id from review ,real_app_category where review.app_id =
     real_app_category.app_id) union all (select app.developer_id, real_app_category.category_id from app,
     real_app_category where app.id = real_app_category.app_id)), user_category_score as (select user_id, category_id,
     count(*) as score from user_category group by user_id, category_id), user_category_like as
     ((select category_user.user_id, category_user.category_id from category_user, category_score where category_user
     .category_id = category_score.category_id) union all (select category_developer.developer_id, category_developer.
     category_id from category_developer, category_score where category_developer.category_id = category_score.category_id))
     , user_and_developer as (select id from user union all select id from developer), user_category_likeit as
     (select user_id, category_id, count(*) as likeit from user_category_like group by user_id, category_id)
     select A.id, E.title as category, if(C.likeit is not null,5,0) + if(D.score is not null,if(D.score>5,5,D.score),0)
      as rating from (((user_and_developer A join category_score B) left join user_category_likeit C on A.id =
      C.user_id and B.category_id = C.category_id) left join user_category_score D on A.id = D.user_id and B.category_id =
      D.category_id), category E where (C.likeit is not null or D.score is not null) and B.category_id = E.id''')
cursor.execute(
    '''create or replace view partial_user_item_rating as with id_count as (select id, count(*) as counta
    from user_item_rating group by id) select A.id, A.category, A.rating from user_item_rating A join id_count B where
     A.id = B.id and B.counta >= 12''')

fopen = open('DMA_project2_team%02d_part2_UIR.csv' % team, 'w', encoding='utf-8', newline='')
col_UIR=['user','category','rating']
for col in col_UIR:
    fopen.write(col)
    if col == 'rating':
        fopen.write('\n')
    else:
        fopen.write(',')
writer2 = csv.writer(fopen)
cursor.execute('''SELECT * FROM partial_user_item_rating''')
b = cursor.fetchall()
for i in b:
    writer2.writerow(i)
fopen.close()
print('2-2 clear')
```

R2-3

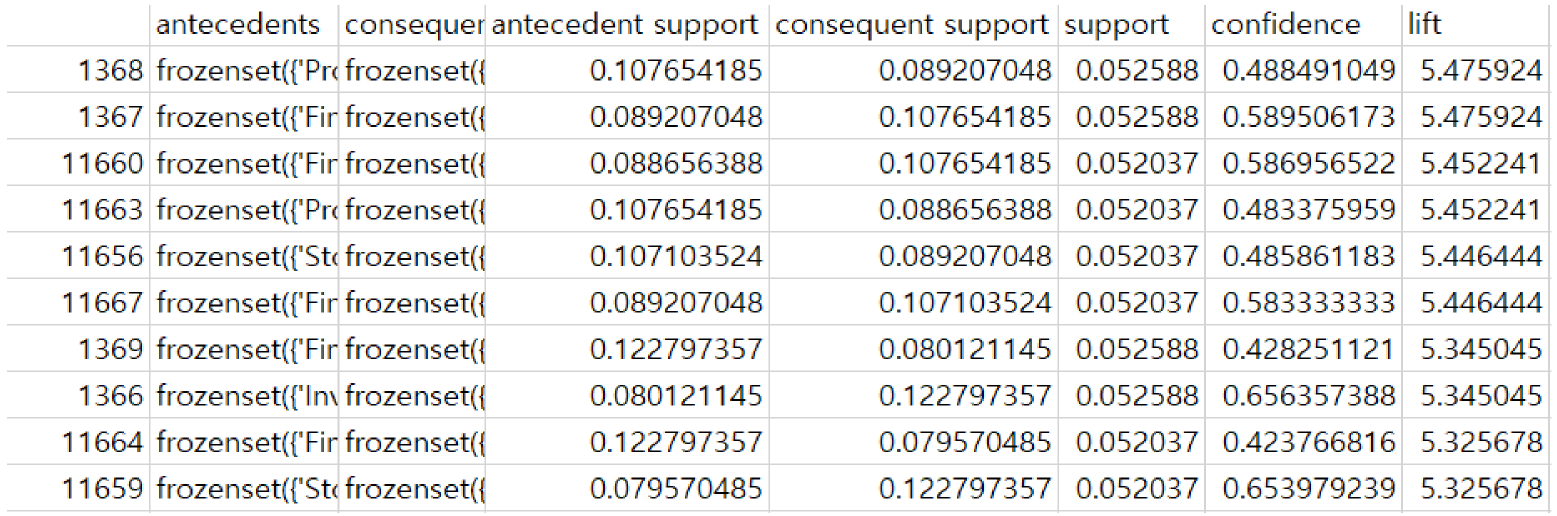


```python
# TODO: Requirement 2-3. MAKE HORIZONTAL VIEW
# file name: DMA_project2_team##_part2_horizontal.pkl
df = pd.read_csv('./DMA_project2_team12_part2_UIR.csv', names=['id', 'category', 'rating'], header=0)
df1 = df.pivot_table(index='id', columns='category', values='rating')
for i in range(2, 11):
    df1 = df1.replace(i, 1)
df1 = df1.fillna(0)
df1.to_pickle('DMA_project2_team12_part2_horizontal.pkl')
```

R2-4

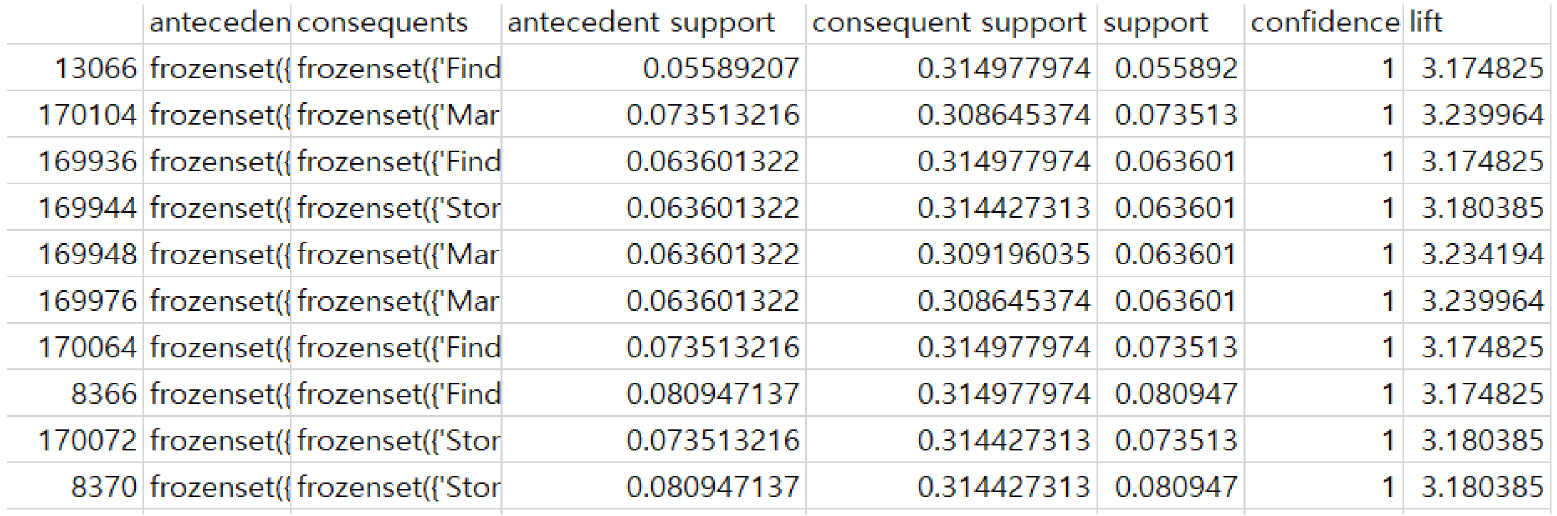


```
# TODO: Requirement 2-4. ASSOCIATION ANALYSIS
# filename: DMA_project2_team##_part2_association.pkl (pandas dataframe)
frequent_itemsets = apriori(df1, min_support=0.05, use_colnames=True)
rules = association_rules(frequent_itemsets, metric='lift', min_threshold=2)
rules.to_pickle('DMA_project2_team12_part2_association.pkl')
cursor.close()

print('2-3 and 2-4 clear')
```



	antecedents	consequer	antecedent support	consequent support	support	confidence	lift
1368	frozenset({'Pr	frozenset({	0.107654185	0.089207048	0.052588	0.488491049	5.475924
1367	frozenset({'Fir	frozenset({	0.089207048	0.107654185	0.052588	0.589506173	5.475924
11660	frozenset({'Fir	frozenset({	0.088656388	0.107654185	0.052037	0.586956522	5.452241
11663	frozenset({'Pr	frozenset({	0.107654185	0.088656388	0.052037	0.483375959	5.452241
11656	frozenset({'St	frozenset({	0.107103524	0.089207048	0.052037	0.485861183	5.446444
11667	frozenset({'Fir	frozenset({	0.089207048	0.107103524	0.052037	0.583333333	5.446444
1369	frozenset({'Fir	frozenset({	0.122797357	0.080121145	0.052588	0.428251121	5.345045
1366	frozenset({'Inv	frozenset({	0.080121145	0.122797357	0.052588	0.656357388	5.345045
11664	frozenset({'Fir	frozenset({	0.122797357	0.079570485	0.052037	0.423766816	5.325678
11659	frozenset({'St	frozenset({	0.079570485	0.122797357	0.052037	0.653979239	5.325678

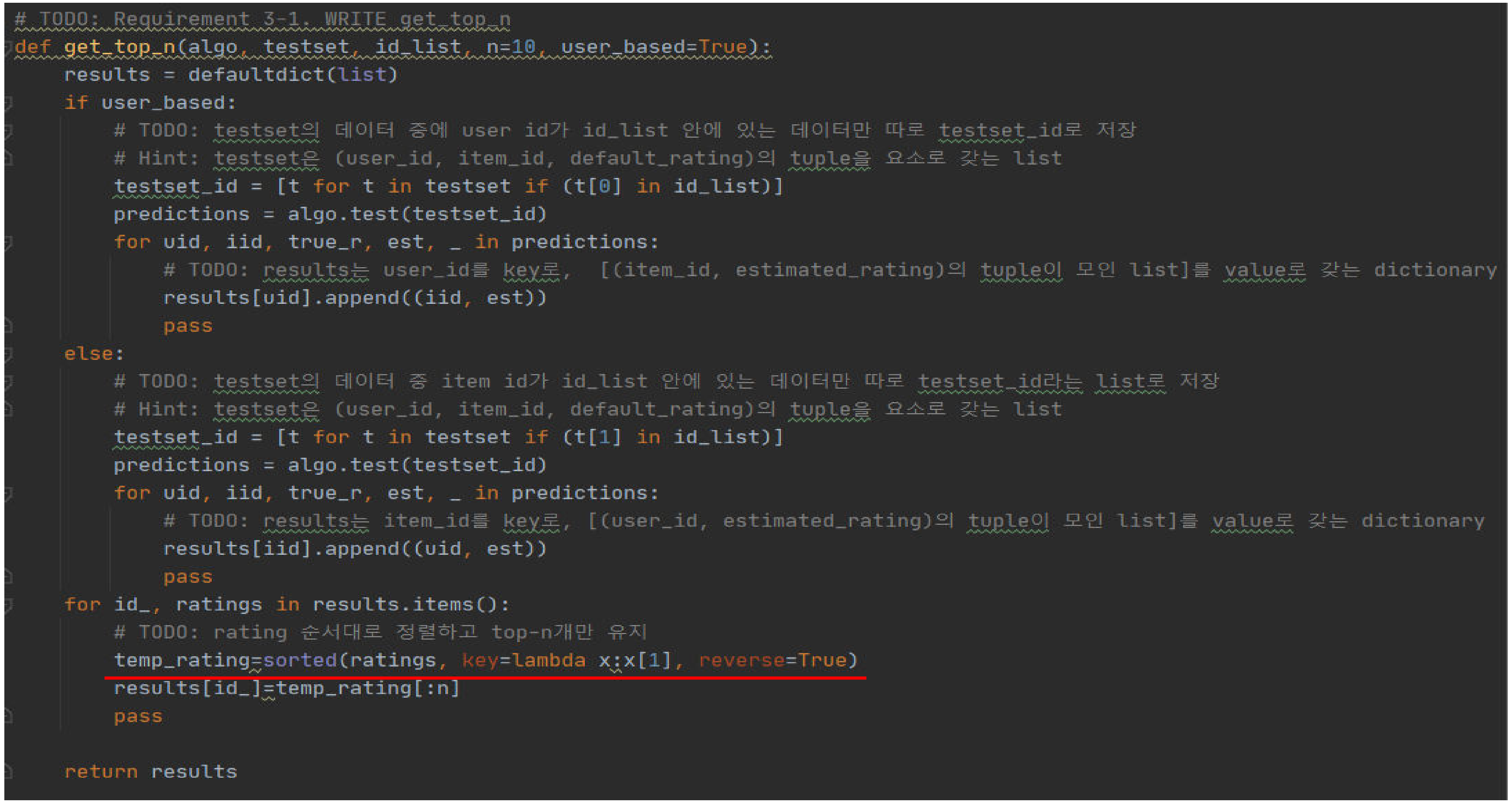
Lift 기준 상위 10개



	anteceden	consequents	antecedent support	consequent support	support	confidence	lift
13066	frozenset({	frozenset({'Find	0.05589207	0.314977974	0.055892	1	3.174825
170104	frozenset({	frozenset({'Mar	0.073513216	0.308645374	0.073513	1	3.239964
169936	frozenset({	frozenset({'Find	0.063601322	0.314977974	0.063601	1	3.174825
169944	frozenset({	frozenset({'Stor	0.063601322	0.314427313	0.063601	1	3.180385
169948	frozenset({	frozenset({'Mar	0.063601322	0.309196035	0.063601	1	3.234194
169976	frozenset({	frozenset({'Mar	0.063601322	0.308645374	0.063601	1	3.239964
170064	frozenset({	frozenset({'Find	0.073513216	0.314977974	0.073513	1	3.174825
8366	frozenset({	frozenset({'Find	0.080947137	0.314977974	0.080947	1	3.174825
170072	frozenset({	frozenset({'Stor	0.073513216	0.314427313	0.073513	1	3.180385
8370	frozenset({	frozenset({'Stor	0.080947137	0.314427313	0.080947	1	3.180385

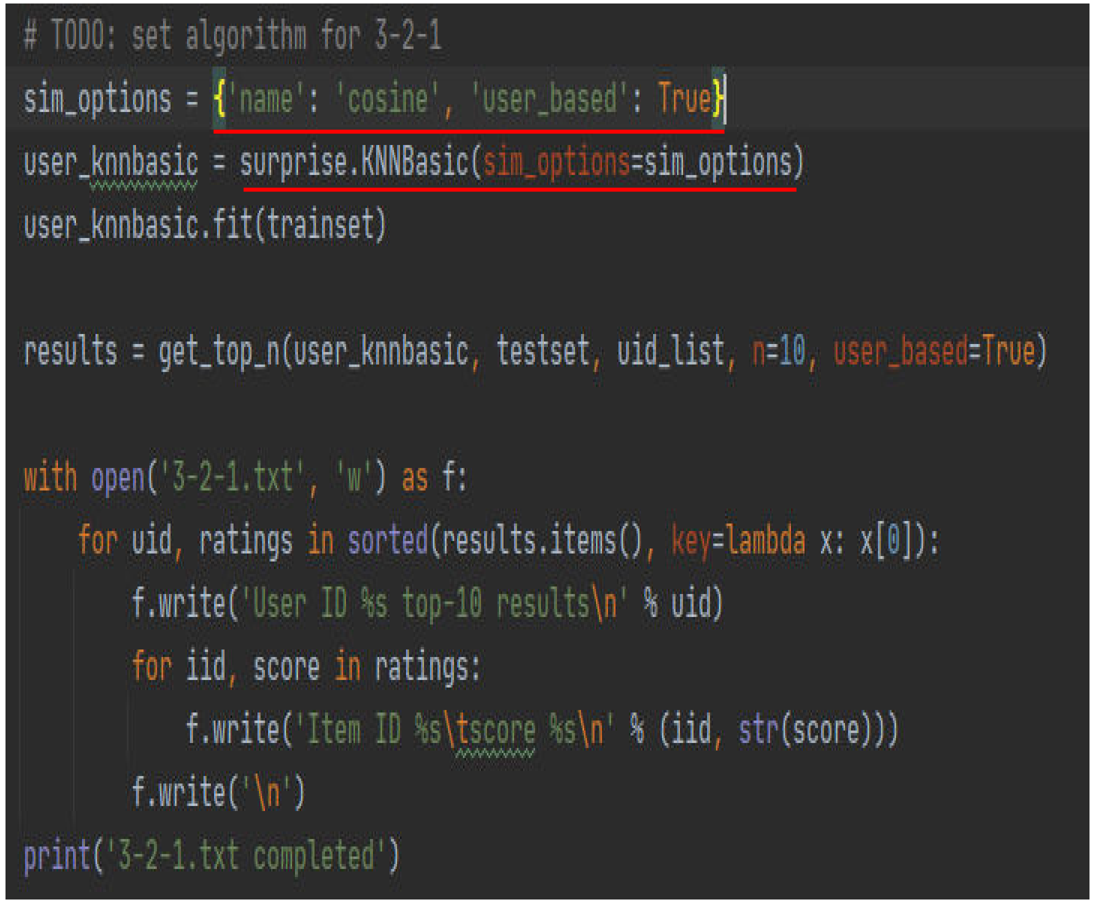
Confidence 기준 상위 10개

R3-1



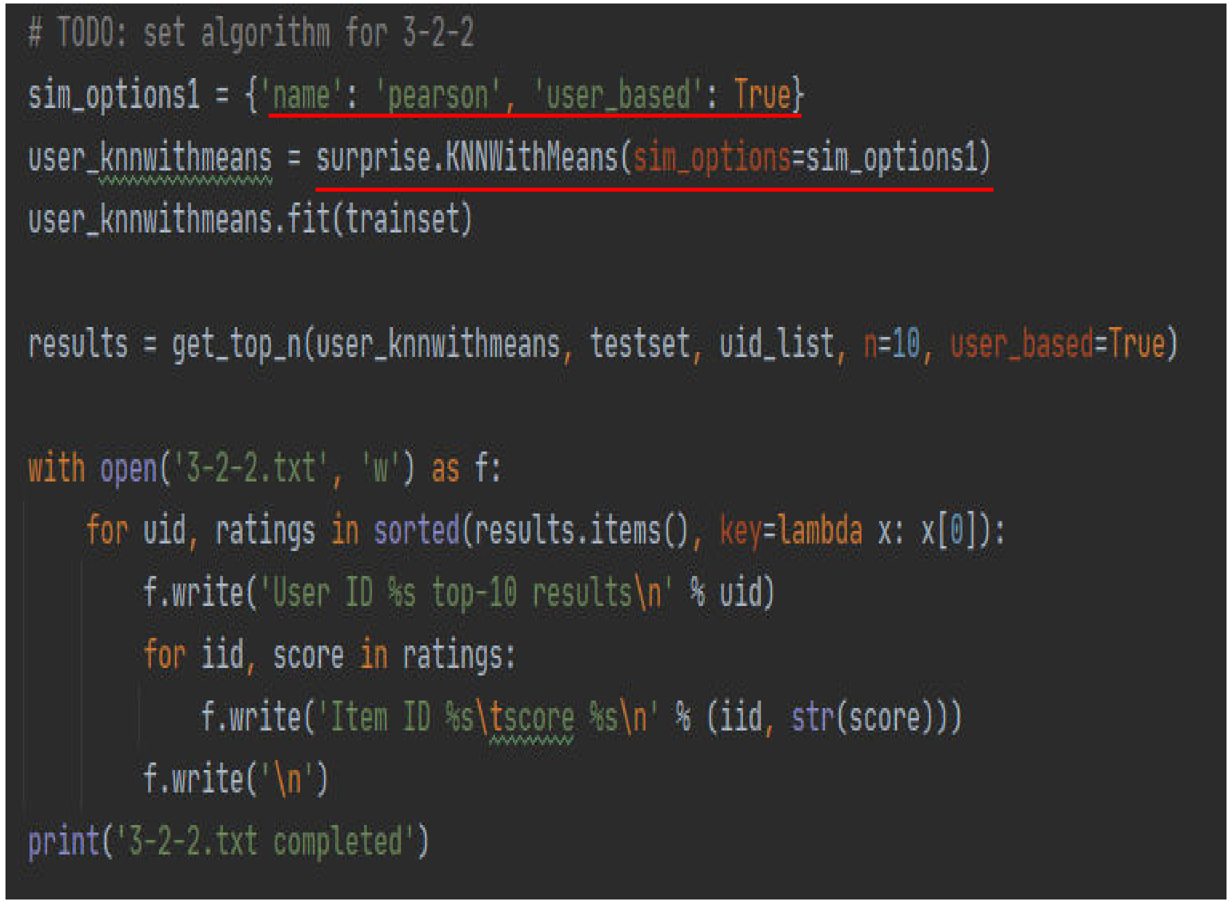
```python
# TODO: Requirement 3-1. WRITE get_top_n
def get_top_n(algo, testset, id_list, n=10, user_based=True):
    results = defaultdict(list)
    if user_based:
        # TODO: testset의 데이터 중에 user id가 id_list 안에 있는 데이터만 따로 testset_id로 저장
        # Hint: testset은 (user_id, item_id, default_rating)의 tuple을 요소로 갖는 list
        testset_id = [t for t in testset if (t[0] in id_list)]
        predictions = algo.test(testset_id)
        for uid, iid, true_r, est, _ in predictions:
            # TODO: results는 user_id를 key로,  [(item_id, estimated_rating)의 tuple이 모인 list]를 value로 갖는 dictionary
            results[uid].append((iid, est))
            pass
    else:
        # TODO: testset의 데이터 중 item id가 id_list 안에 있는 데이터만 따로 testset_id라는 list로 저장
        # Hint: testset은 (user_id, item_id, default_rating)의 tuple을 요소로 갖는 list
        testset_id = [t for t in testset if (t[1] in id_list)]
        predictions = algo.test(testset_id)
        for uid, iid, true_r, est, _ in predictions:
            # TODO: results는 item_id를 key로, [(user_id, estimated_rating)의 tuple이 모인 list]를 value로 갖는 dictionary
            results[iid].append((uid, est))
            pass
    for id_, ratings in results.items():
        # TODO: rating 순서대로 정렬하고 top-n개만 유지
        temp_rating=sorted(ratings, key=lambda x:x[1], reverse=True)
        results[id_]=temp_rating[:n]
        pass

    return results
```

R3-2-1



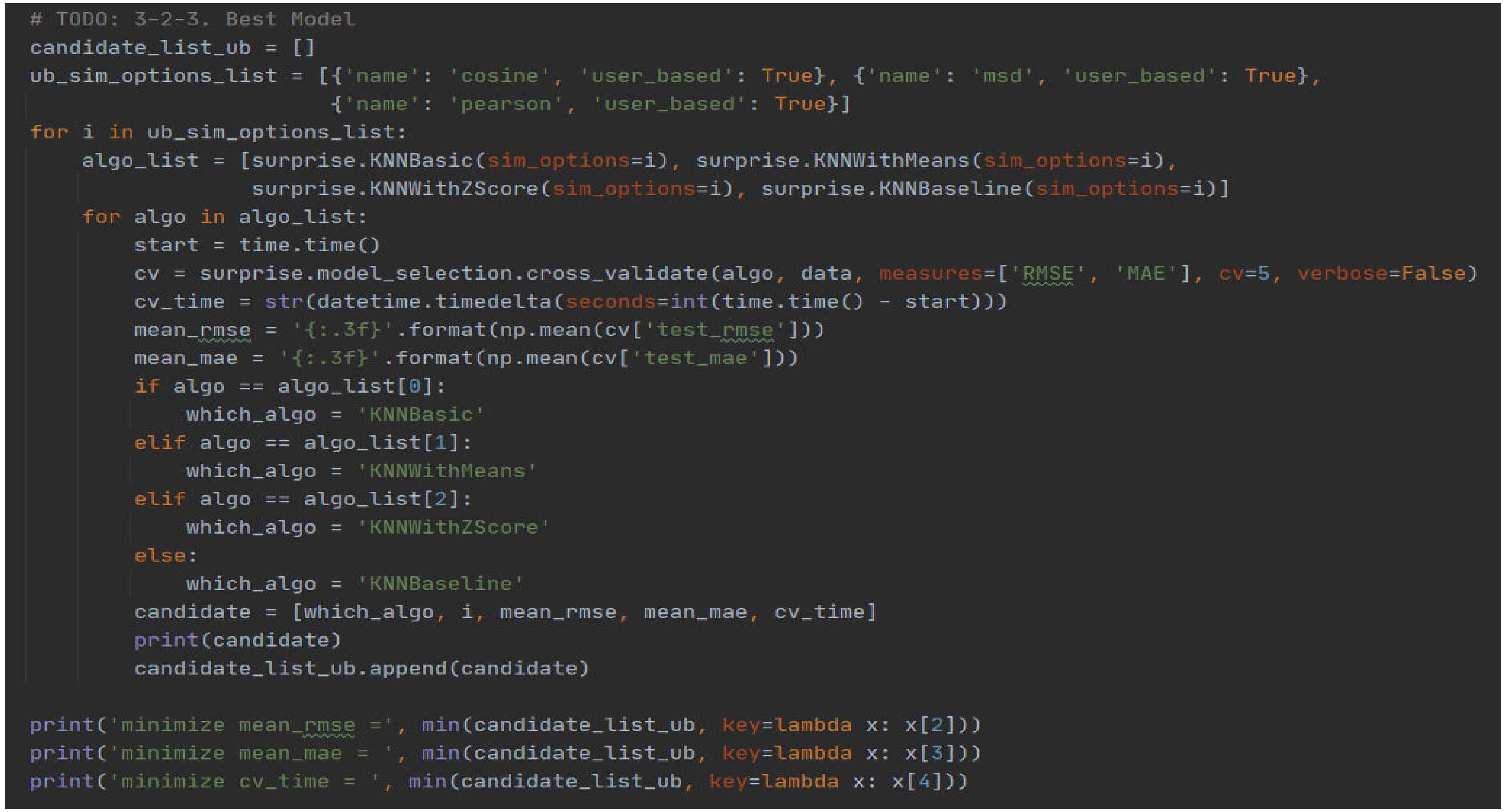
```
# TODO: set algorithm for 3-2-1
sim_options = {'name': 'cosine', 'user_based': True}
user_knnbasic = surprise.KNNBasic(sim_options=sim_options)
user_knnbasic.fit(trainset)

results = get_top_n(user_knnbasic, testset, uid_list, n=10, user_based=True)

with open('3-2-1.txt', 'w') as f:
    for uid, ratings in sorted(results.items(), key=lambda x: x[0]):
        f.write('User ID %s top-10 results\n' % uid)
        for iid, score in ratings:
            f.write('Item ID %s\tscore %s\n' % (iid, str(score)))
        f.write('\n')
print('3-2-1.txt completed')
```

R3-2-2



```
# TODO: set algorithm for 3-2-2
sim_options1 = {'name': 'pearson', 'user_based': True}
user_knnwithmeans = surprise.KNNWithMeans(sim_options=sim_options1)
user_knnwithmeans.fit(trainset)

results = get_top_n(user_knnwithmeans, testset, uid_list, n=10, user_based=True)

with open('3-2-2.txt', 'w') as f:
    for uid, ratings in sorted(results.items(), key=lambda x: x[0]):
        f.write('User ID %s top-10 results\n' % uid)
        for iid, score in ratings:
            f.write('Item ID %s\tscore %s\n' % (iid, str(score)))
        f.write('\n')
print('3-2-2.txt completed')
```

R 3-2-3



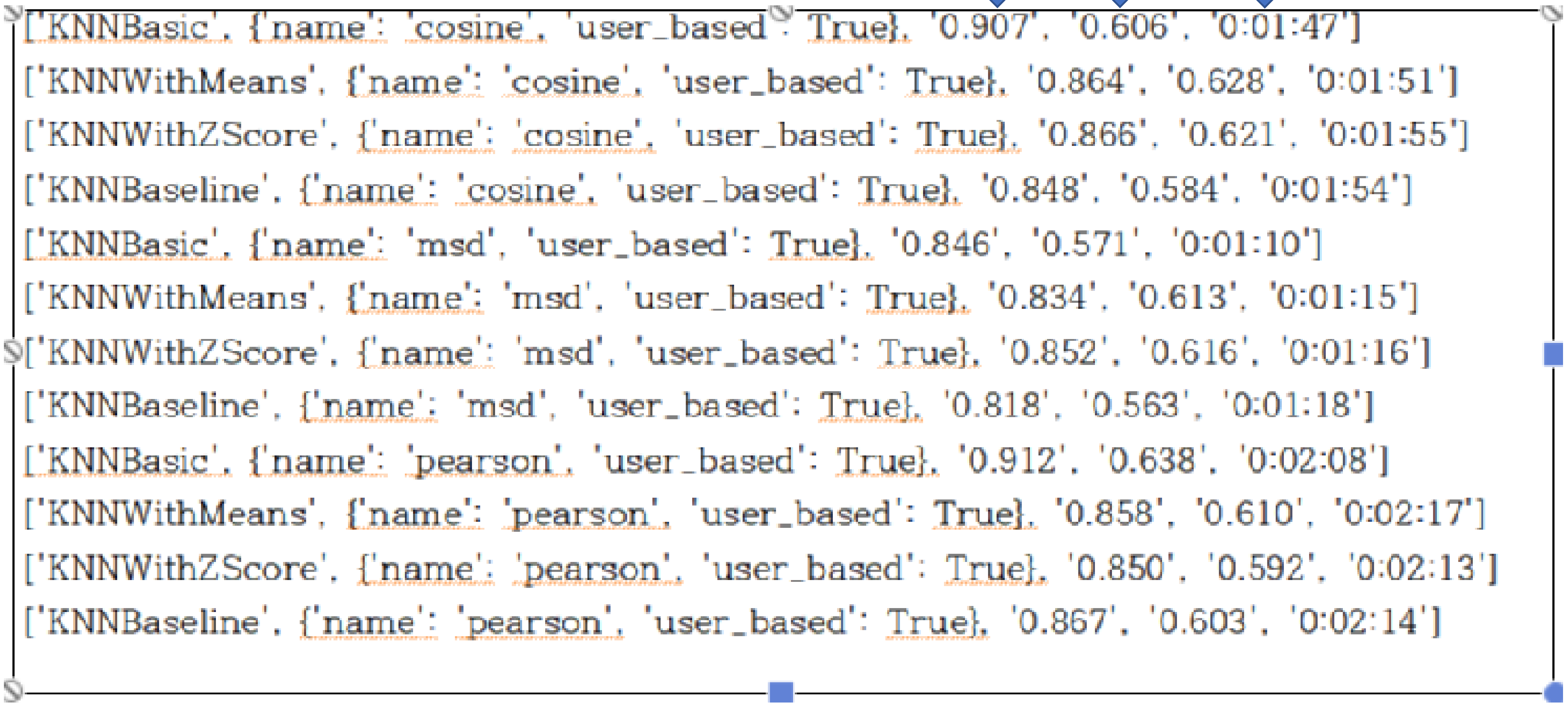
```python
# TODO: 3-2-3. Best Model
candidate_list_ub = []
ub_sim_options_list = [{'name': 'cosine', 'user_based': True}, {'name': 'msd', 'user_based': True},
                       {'name': 'pearson', 'user_based': True}]
for i in ub_sim_options_list:
    algo_list = [surprise.KNNBasic(sim_options=i), surprise.KNNWithMeans(sim_options=i),
                 surprise.KNNWithZScore(sim_options=i), surprise.KNNBaseline(sim_options=i)]
    for algo in algo_list:
        start = time.time()
        cv = surprise.model_selection.cross_validate(algo, data, measures=['RMSE', 'MAE'], cv=5, verbose=False)
        cv_time = str(datetime.timedelta(seconds=int(time.time() - start)))
        mean_rmse = '{:.3f}'.format(np.mean(cv['test_rmse']))
        mean_mae = '{:.3f}'.format(np.mean(cv['test_mae']))
        if algo == algo_list[0]:
            which_algo = 'KNNBasic'
        elif algo == algo_list[1]:
            which_algo = 'KNNWithMeans'
        elif algo == algo_list[2]:
            which_algo = 'KNNWithZScore'
        else:
            which_algo = 'KNNBaseline'
        candidate = [which_algo, i, mean_rmse, mean_mae, cv_time]
        print(candidate)
        candidate_list_ub.append(candidate)

print('minimize mean_rmse =', min(candidate_list_ub, key=lambda x: x[2]))
print('minimize mean_mae = ', min(candidate_list_ub, key=lambda x: x[3]))
print('minimize cv_time = ', min(candidate_list_ub, key=lambda x: x[4]))
```

R3-2-3 고려사항

RMSE / MAE / time 세가지 경우에 대해 살펴보았다.
k: (max)number of nearest neighbors
k가 너무 작으면 noise points에 민감해지고 k가 너무 크면 다른 class에 속한 points들을 포함할 가능성이 커지는 점을 고려했다. (ch.8 slide 50)
실제로 k를 증가시켜 (k=150 등) cross validation을 해보면, 같은 algorithm과 name 하에서 k만 변화시켰을 때 RMSE와 MAE에 큰 영향을 주지 못함을 확인할 수 있었다.
따라서, k=40 (default value) 로 특별히 변화시키지 않고 algorithm의 종류와 유사도(name)의 종류만 변화시켜 성능을 비교 및 평가하였다.
사용한 algorithm의 종류: KNNBasic, KNNWithMeans, KNNWithZScore, KNNBaseline
사용한 유사도(name)의 종류: cosine, msd, pearson
따라서, 총 12개 model에 대해 성능을 비교 및 평가하였다.

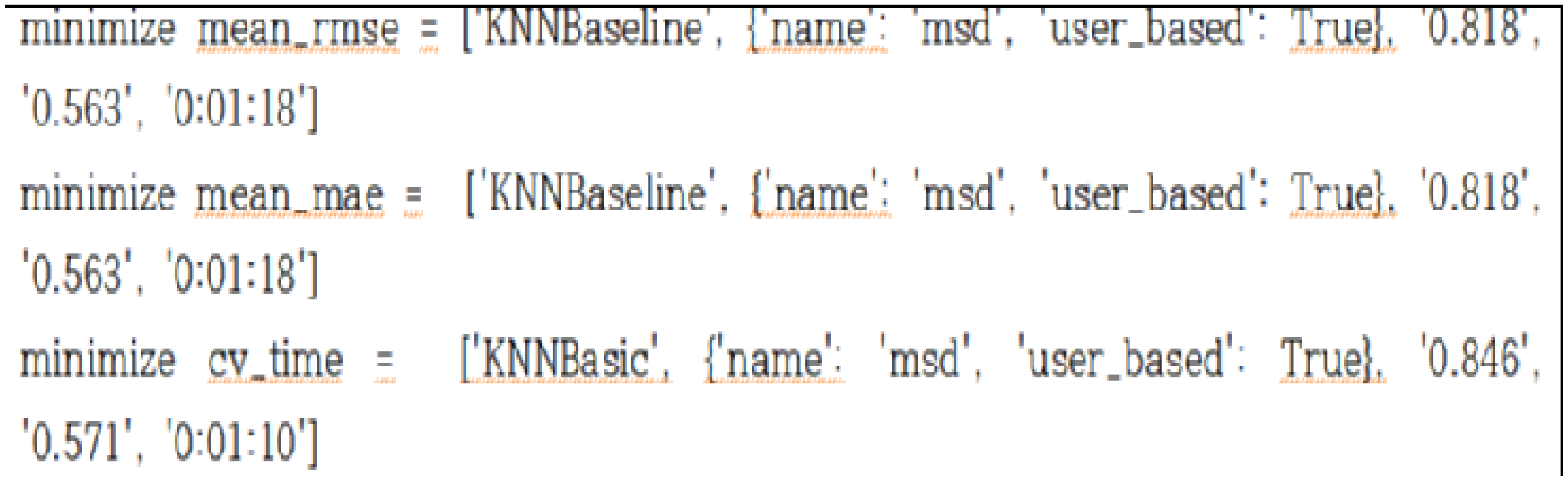
R3-2-3 결과

rmse mae time



```
['KNNBasic', {'name': 'cosine', 'user_based': True}, '0.907', '0.606', '0:01:47']
['KNNWithMeans', {'name': 'cosine', 'user_based': True}, '0.864', '0.628', '0:01:51']
['KNNWithZScore', {'name': 'cosine', 'user_based': True}, '0.866', '0.621', '0:01:55']
['KNNBaseline', {'name': 'cosine', 'user_based': True}, '0.848', '0.584', '0:01:54']
['KNNBasic', {'name': 'msd', 'user_based': True}, '0.846', '0.571', '0:01:10']
['KNNWithMeans', {'name': 'msd', 'user_based': True}, '0.834', '0.613', '0:01:15']
['KNNWithZScore', {'name': 'msd', 'user_based': True}, '0.852', '0.616', '0:01:16']
['KNNBaseline', {'name': 'msd', 'user_based': True}, '0.818', '0.563', '0:01:18']
['KNNBasic', {'name': 'pearson', 'user_based': True}, '0.912', '0.638', '0:02:08']
['KNNWithMeans', {'name': 'pearson', 'user_based': True}, '0.858', '0.610', '0:02:17']
['KNNWithZScore', {'name': 'pearson', 'user_based': True}, '0.850', '0.592', '0:02:13']
['KNNBaseline', {'name': 'pearson', 'user_based': True}, '0.867', '0.603', '0:02:14']
```

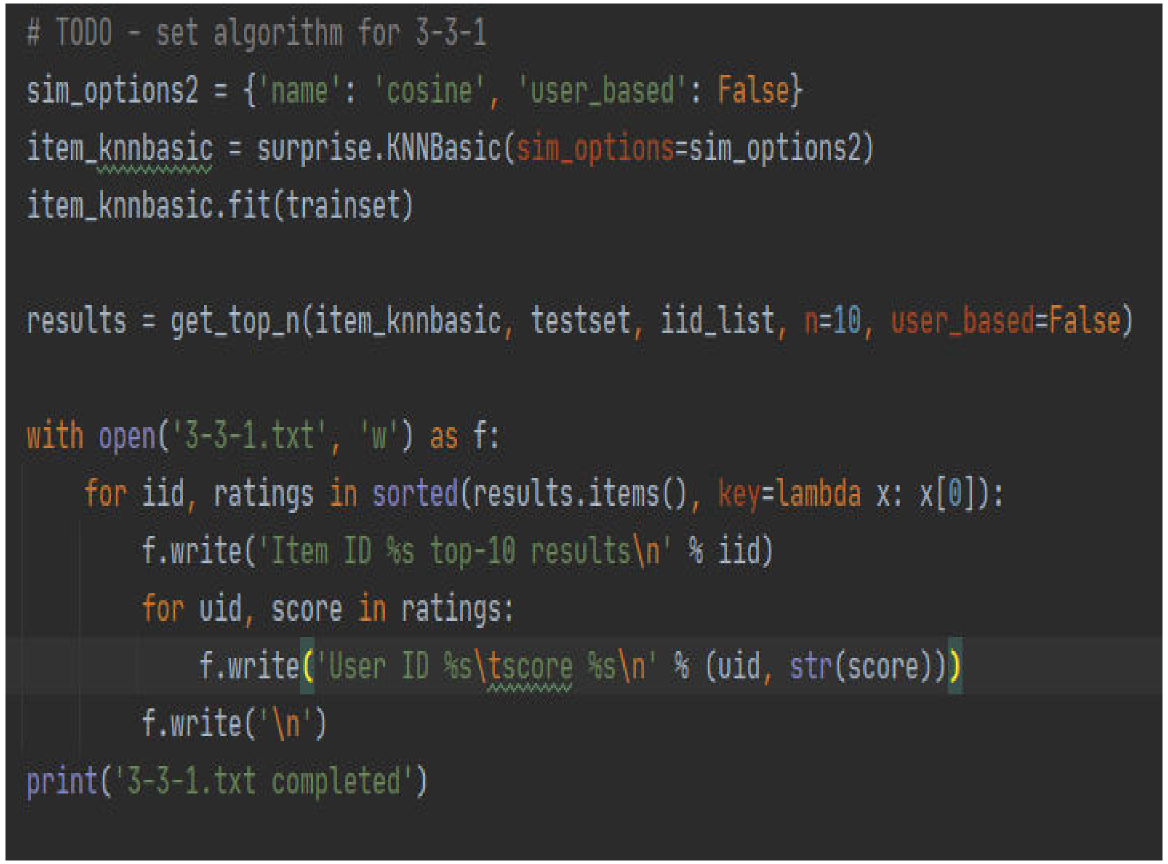
R3-2-3 결과 (최종결과)



```
minimize mean_rmse = ['KNNBaseline', {'name': 'msd', 'user_based': True}, '0.818',
'0.563', '0:01:18']
minimize mean_mae =  ['KNNBaseline', {'name': 'msd', 'user_based': True}, '0.818',
'0.563', '0:01:18']
minimize cv_time =   ['KNNBasic', {'name': 'msd', 'user_based': True}, '0.846',
'0.571', '0:01:10']
```

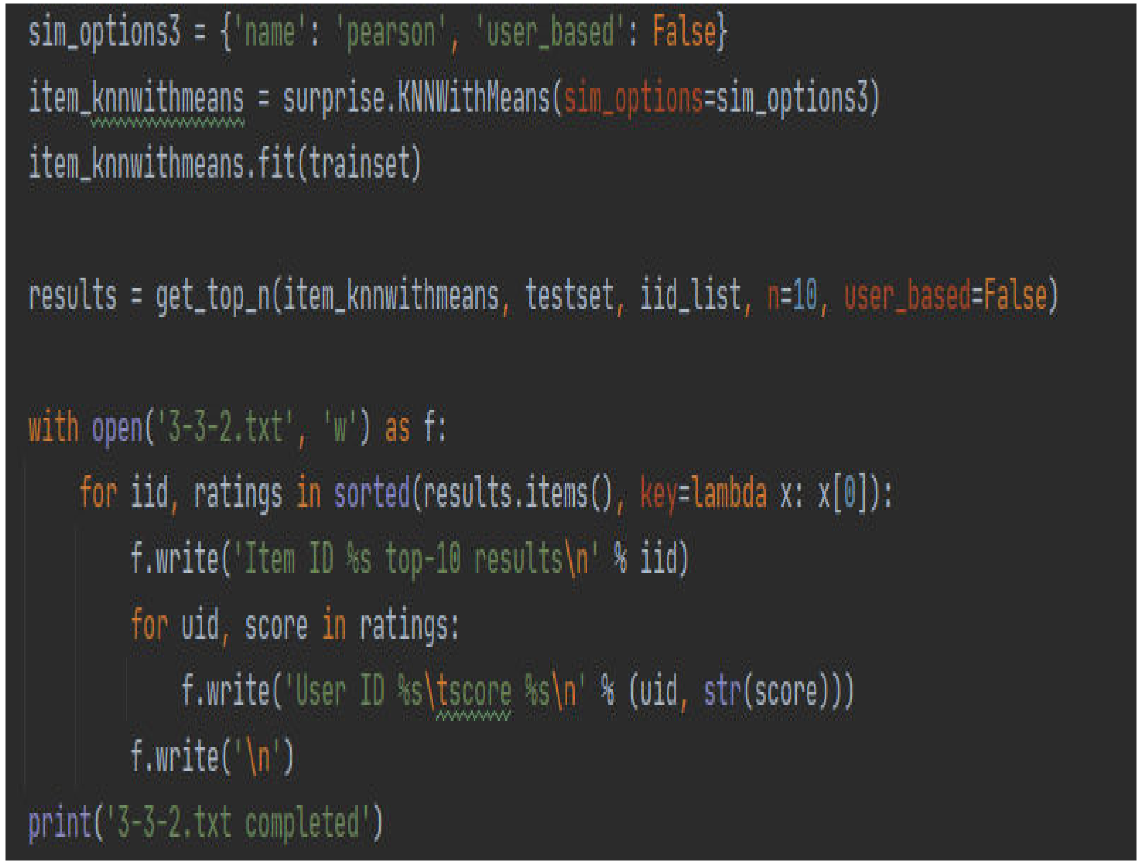
따라서, 가장 좋은 성능을 보이는 모델 best_model_ub은 (RMSE 기준)
best_model_ub = surprise.KNNBaseline(sim_options = {'name: 'msd', 'user_based': True})
이다.

R3-3-1



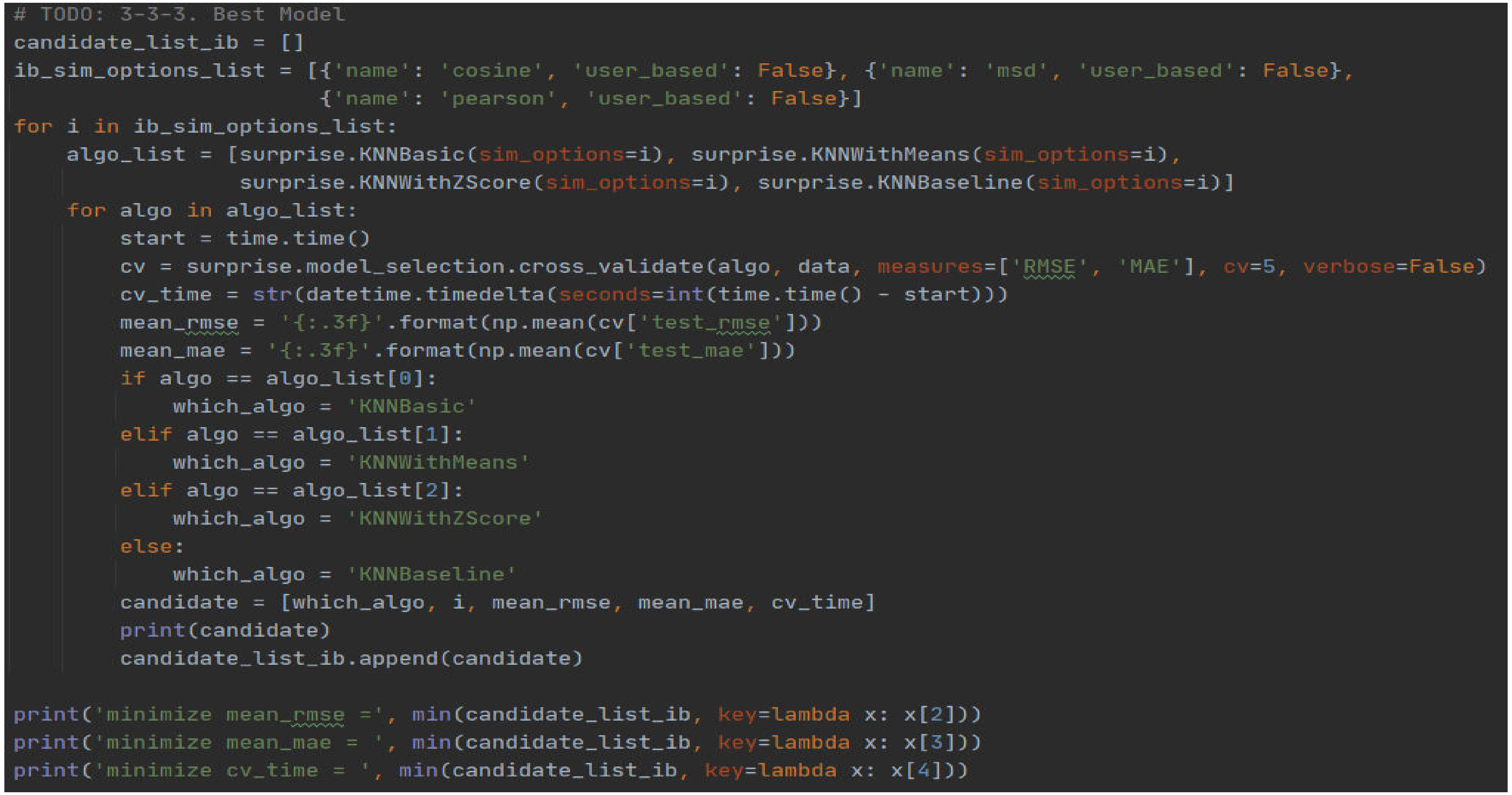
```python
# TODO - set algorithm for 3-3-1
sim_options2 = {'name': 'cosine', 'user_based': False}
item_knnbasic = surprise.KNNBasic(sim_options=sim_options2)
item_knnbasic.fit(trainset)

results = get_top_n(item_knnbasic, testset, iid_list, n=10, user_based=False)

with open('3-3-1.txt', 'w') as f:
    for iid, ratings in sorted(results.items(), key=lambda x: x[0]):
        f.write('Item ID %s top-10 results\n' % iid)
        for uid, score in ratings:
            f.write('User ID %s\tscore %s\n' % (uid, str(score)))
        f.write('\n')
print('3-3-1.txt completed')
```

R3-3-2



```python
sim_options3 = {'name': 'pearson', 'user_based': False}
item_knnwithmeans = surprise.KNNWithMeans(sim_options=sim_options3)
item_knnwithmeans.fit(trainset)

results = get_top_n(item_knnwithmeans, testset, iid_list, n=10, user_based=False)

with open('3-3-2.txt', 'w') as f:
    for iid, ratings in sorted(results.items(), key=lambda x: x[0]):
        f.write('Item ID %s top-10 results\n' % iid)
        for uid, score in ratings:
            f.write('User ID %s\tscore %s\n' % (uid, str(score)))
        f.write('\n')
print('3-3-2.txt completed')
```

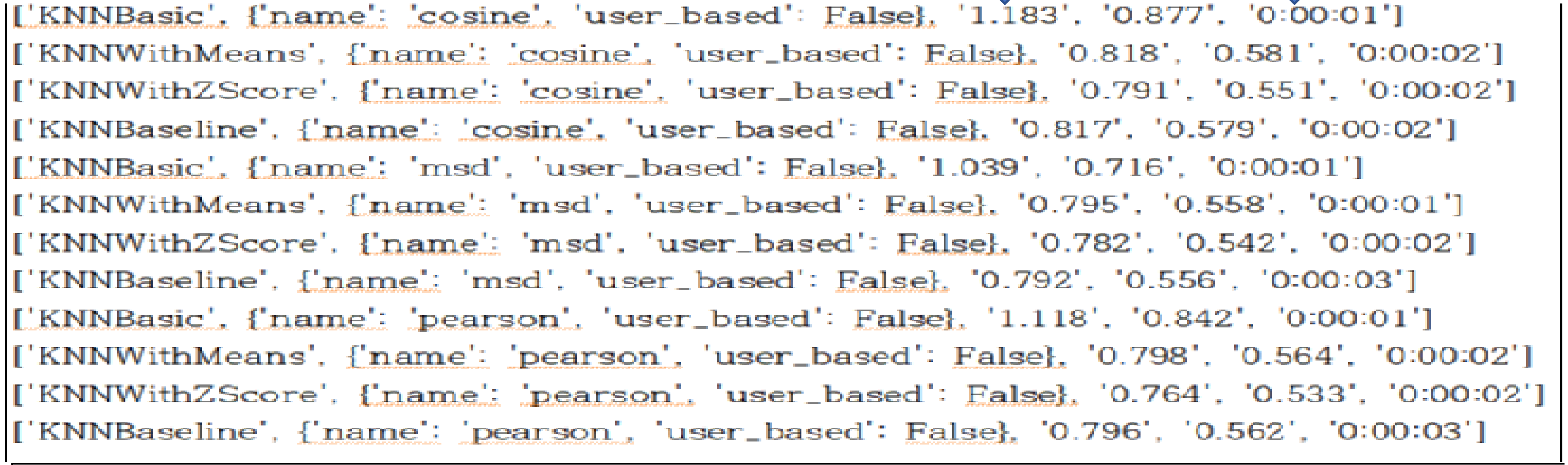
R3-3-3



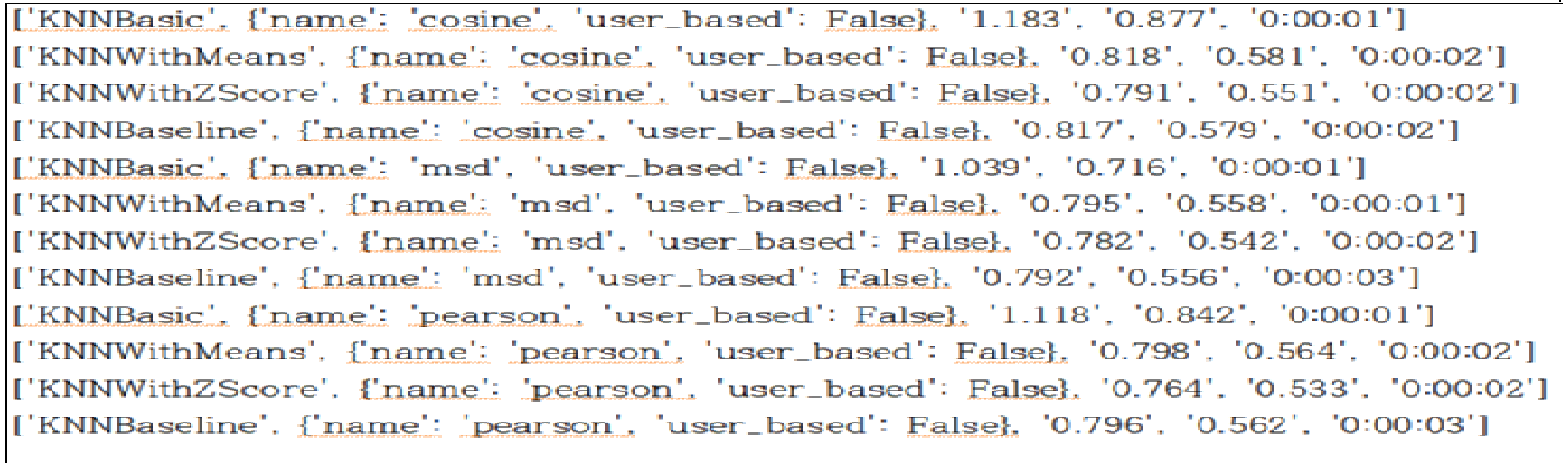
```python
# TODO: 3-3-3. Best Model
candidate_list_ib = []
ib_sim_options_list = [{'name': 'cosine', 'user_based': False}, {'name': 'msd', 'user_based': False},
                       {'name': 'pearson', 'user_based': False}]
for i in ib_sim_options_list:
    algo_list = [surprise.KNNBasic(sim_options=i), surprise.KNNWithMeans(sim_options=i),
                 surprise.KNNWithZScore(sim_options=i), surprise.KNNBaseline(sim_options=i)]
    for algo in algo_list:
        start = time.time()
        cv = surprise.model_selection.cross_validate(algo, data, measures=['RMSE', 'MAE'], cv=5, verbose=False)
        cv_time = str(datetime.timedelta(seconds=int(time.time() - start)))
        mean_rmse = '{:.3f}'.format(np.mean(cv['test_rmse']))
        mean_mae = '{:.3f}'.format(np.mean(cv['test_mae']))
        if algo == algo_list[0]:
            which_algo = 'KNNBasic'
        elif algo == algo_list[1]:
            which_algo = 'KNNWithMeans'
        elif algo == algo_list[2]:
            which_algo = 'KNNWithZScore'
        else:
            which_algo = 'KNNBaseline'
        candidate = [which_algo, i, mean_rmse, mean_mae, cv_time]
        print(candidate)
        candidate_list_ib.append(candidate)

print('minimize mean_rmse =', min(candidate_list_ib, key=lambda x: x[2]))
print('minimize mean_mae = ', min(candidate_list_ib, key=lambda x: x[3]))
print('minimize cv_time = ', min(candidate_list_ib, key=lambda x: x[4]))
```

R3-3-3 결과

rmse mae time

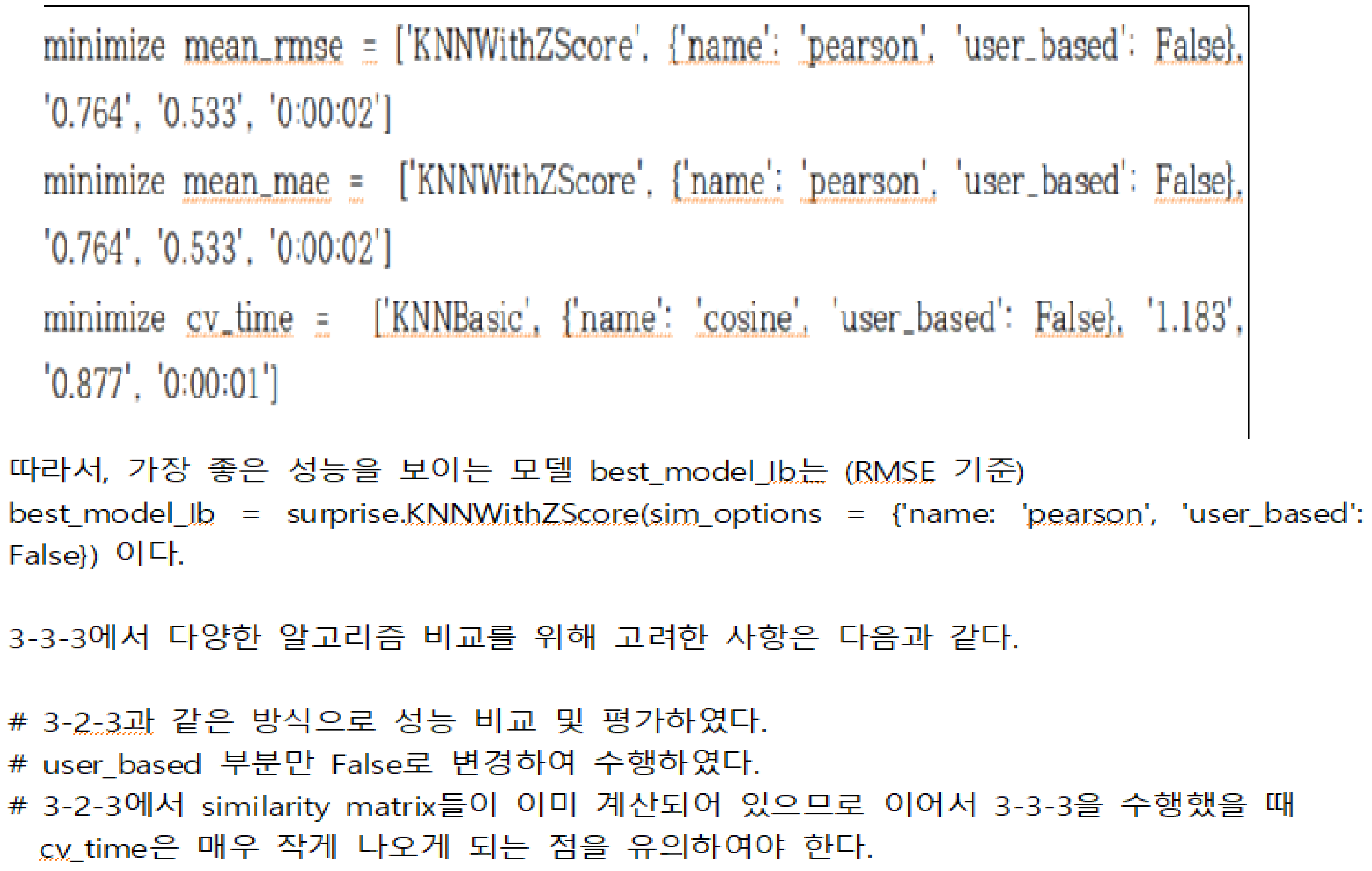


```
['KNNBasic', {'name': 'cosine', 'user_based': False}, '1.183', '0.877', '0:00:01']
['KNNWithMeans', {'name': 'cosine', 'user_based': False}, '0.818', '0.581', '0:00:02']
['KNNWithZScore', {'name': 'cosine', 'user_based': False}, '0.791', '0.551', '0:00:02']
['KNNBaseline', {'name': 'cosine', 'user_based': False}, '0.817', '0.579', '0:00:02']
['KNNBasic', {'name': 'msd', 'user_based': False}, '1.039', '0.716', '0:00:01']
['KNNWithMeans', {'name': 'msd', 'user_based': False}, '0.795', '0.558', '0:00:01']
['KNNWithZScore', {'name': 'msd', 'user_based': False}, '0.782', '0.542', '0:00:02']
['KNNBaseline', {'name': 'msd', 'user_based': False}, '0.792', '0.556', '0:00:03']
['KNNBasic', {'name': 'pearson', 'user_based': False}, '1.118', '0.842', '0:00:01']
['KNNWithMeans', {'name': 'pearson', 'user_based': False}, '0.798', '0.564', '0:00:02']
['KNNWithZScore', {'name': 'pearson', 'user_based': False}, '0.764', '0.533', '0:00:02']
['KNNBaseline', {'name': 'pearson', 'user_based': False}, '0.796', '0.562', '0:00:03']
```



```
['KNNBasic', {'name': 'cosine', 'user_based': False}, '1.183', '0.877', '0:00:01']
['KNNWithMeans', {'name': 'cosine', 'user_based': False}, '0.818', '0.581', '0:00:02']
['KNNWithZScore', {'name': 'cosine', 'user_based': False}, '0.791', '0.551', '0:00:02']
['KNNBaseline', {'name': 'cosine', 'user_based': False}, '0.817', '0.579', '0:00:02']
['KNNBasic', {'name': 'msd', 'user_based': False}, '1.039', '0.716', '0:00:01']
['KNNWithMeans', {'name': 'msd', 'user_based': False}, '0.795', '0.558', '0:00:01']
['KNNWithZScore', {'name': 'msd', 'user_based': False}, '0.782', '0.542', '0:00:02']
['KNNBaseline', {'name': 'msd', 'user_based': False}, '0.792', '0.556', '0:00:03']
['KNNBasic', {'name': 'pearson', 'user_based': False}, '1.118', '0.842', '0:00:01']
['KNNWithMeans', {'name': 'pearson', 'user_based': False}, '0.798', '0.564', '0:00:02']
['KNNWithZScore', {'name': 'pearson', 'user_based': False}, '0.764', '0.533', '0:00:02']
['KNNBaseline', {'name': 'pearson', 'user_based': False}, '0.796', '0.562', '0:00:03']
```

R3-3-3 결과 (최종결과)



```
minimize mean_rmse = ['KNNWithZScore', {'name': 'pearson', 'user_based': False},
'0.764', '0.533', '0:00:02']
minimize mean_mae = ['KNNWithZScore', {'name': 'pearson', 'user_based': False},
'0.764', '0.533', '0:00:02']
minimize cv_time = ['KNNBasic', {'name': 'cosine', 'user_based': False}, '1.183',
'0.877', '0:00:01']
```

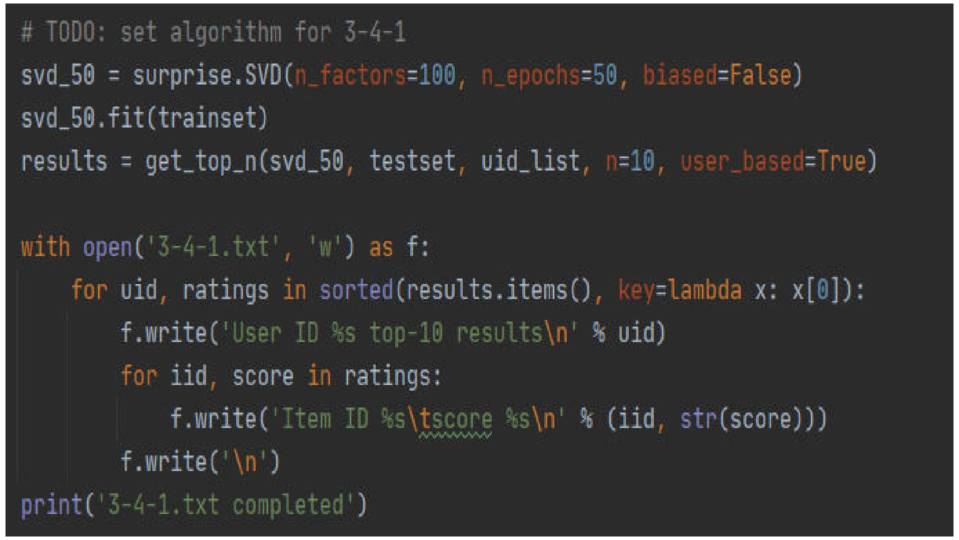
따라서, 가장 좋은 성능을 보이는 모델 best_model_Ib는 (RMSE 기준)
best_model_Ib = surprise.KNNWithZScore(sim_options = {'name: 'pearson', 'user_based': False}) 이다.

3-3-3에서 다양한 알고리즘 비교를 위해 고려한 사항은 다음과 같다.

3-2-3과 같은 방식으로 성능 비교 및 평가하였다.
user_based 부분만 False로 변경하여 수행하였다.
3-2-3에서 similarity matrix들이 이미 계산되어 있으므로 이어서 3-3-3을 수행했을 때
cv_time은 매우 작게 나오게 되는 점을 유의하여야 한다.

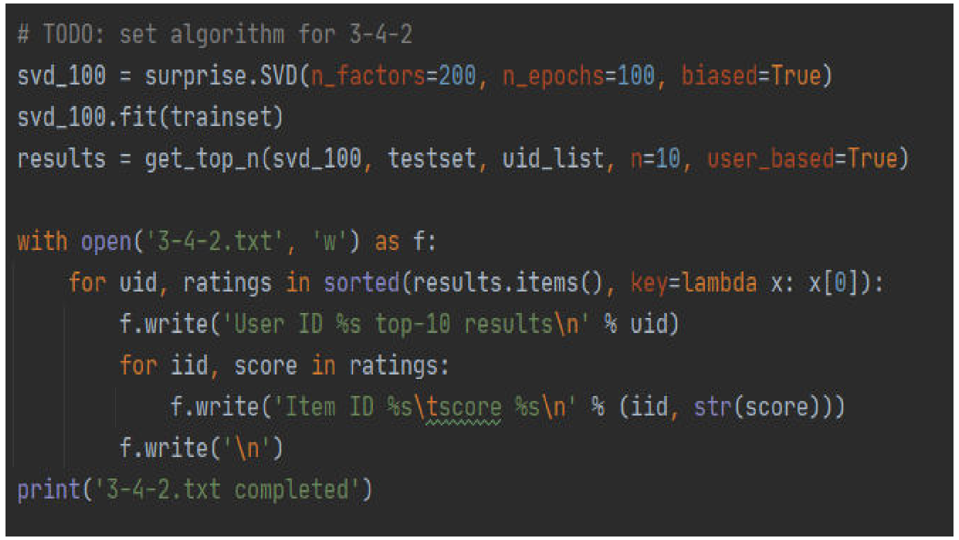
고려사항

R3-4-1



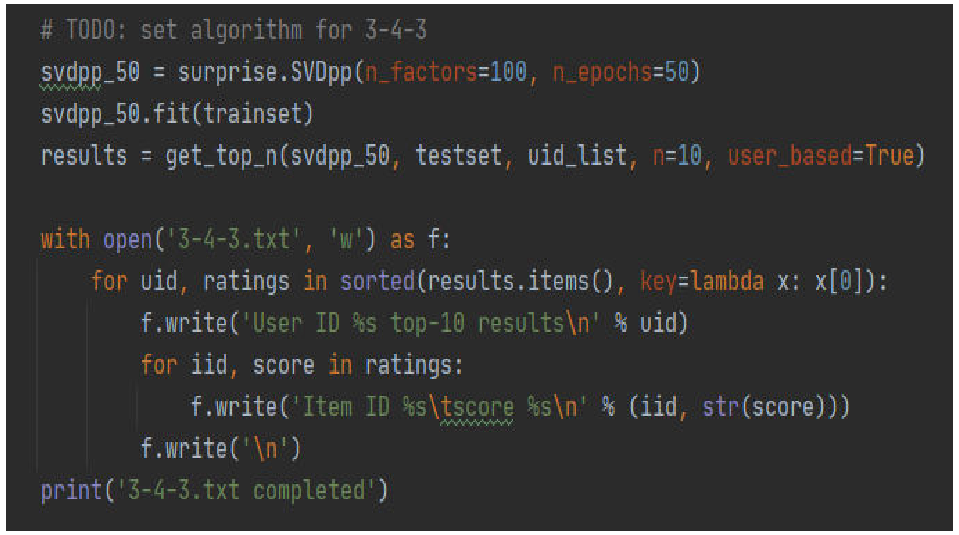
```python
# TODO: set algorithm for 3-4-1
svd_50 = surprise.SVD(n_factors=100, n_epochs=50, biased=False)
svd_50.fit(trainset)
results = get_top_n(svd_50, testset, uid_list, n=10, user_based=True)

with open('3-4-1.txt', 'w') as f:
    for uid, ratings in sorted(results.items(), key=lambda x: x[0]):
        f.write('User ID %s top-10 results\n' % uid)
        for iid, score in ratings:
            f.write('Item ID %s\tscore %s\n' % (iid, str(score)))
        f.write('\n')
print('3-4-1.txt completed')
```

R3-4-2



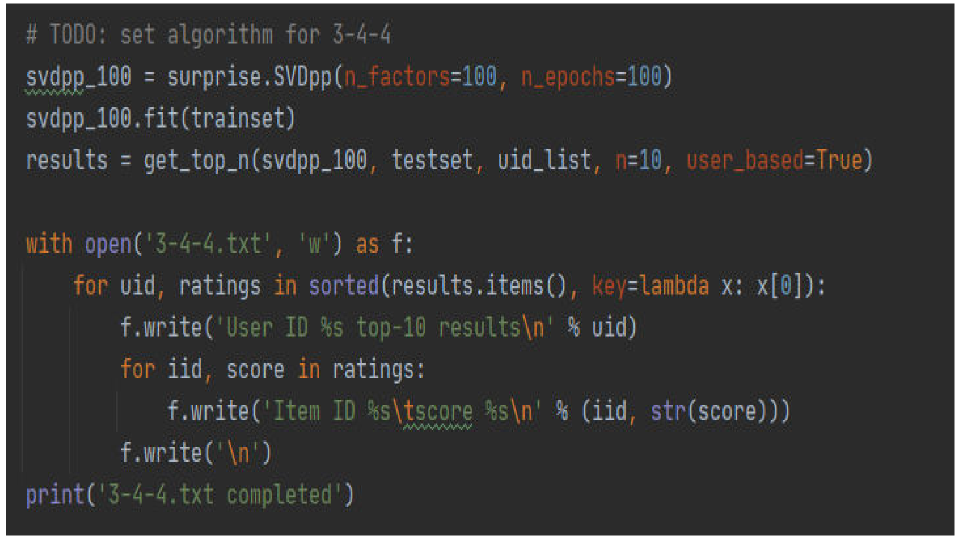
```python
# TODO: set algorithm for 3-4-2
svd_100 = surprise.SVD(n_factors=200, n_epochs=100, biased=True)
svd_100.fit(trainset)
results = get_top_n(svd_100, testset, uid_list, n=10, user_based=True)

with open('3-4-2.txt', 'w') as f:
    for uid, ratings in sorted(results.items(), key=lambda x: x[0]):
        f.write('User ID %s top-10 results\n' % uid)
        for iid, score in ratings:
            f.write('Item ID %s\tscore %s\n' % (iid, str(score)))
        f.write('\n')
print('3-4-2.txt completed')
```

R3-4-3



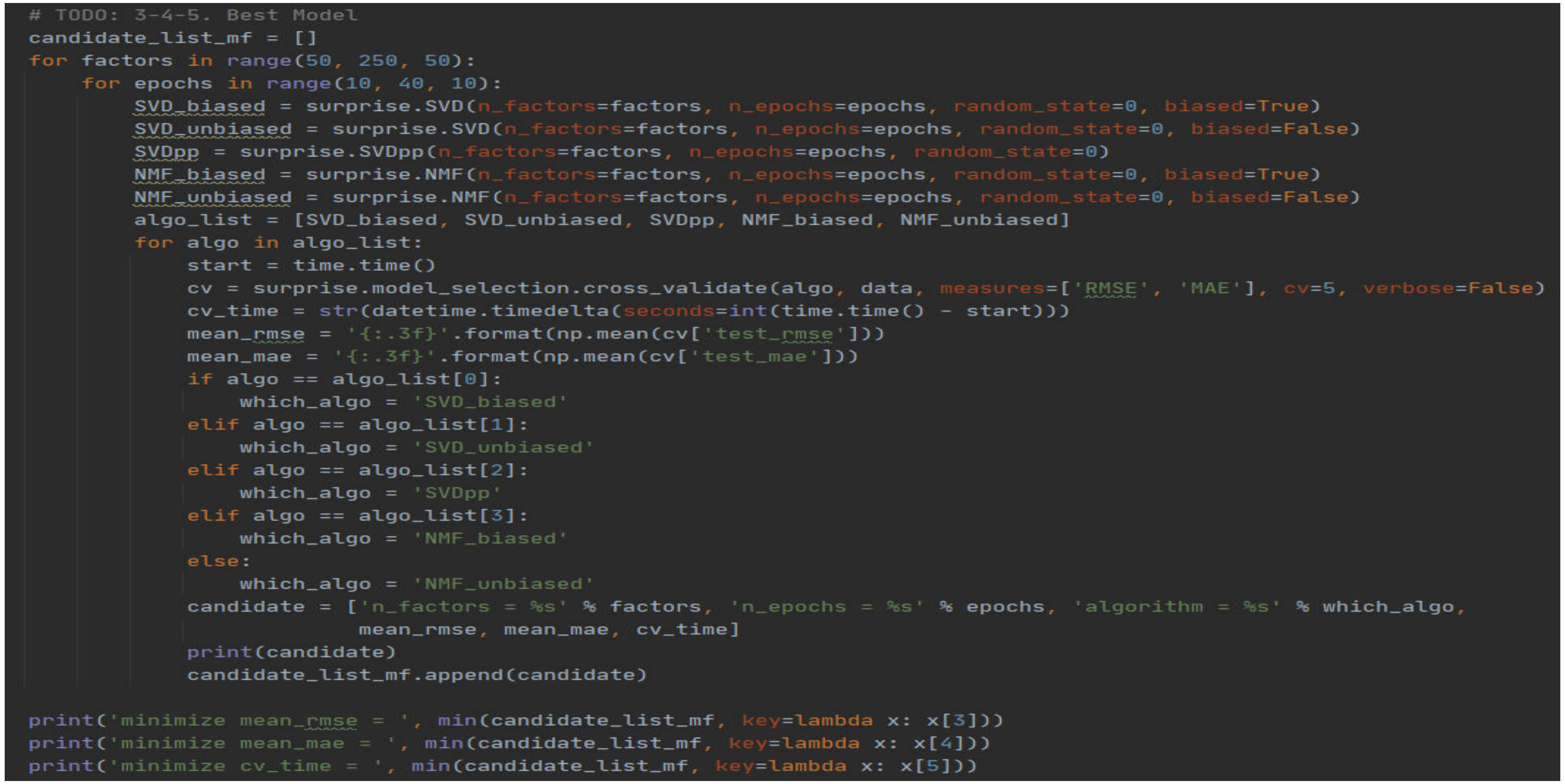
```python
# TODO: set algorithm for 3-4-3
svdpp_50 = surprise.SVDpp(n_factors=100, n_epochs=50)
svdpp_50.fit(trainset)
results = get_top_n(svdpp_50, testset, uid_list, n=10, user_based=True)

with open('3-4-3.txt', 'w') as f:
    for uid, ratings in sorted(results.items(), key=lambda x: x[0]):
        f.write('User ID %s top-10 results\n' % uid)
        for iid, score in ratings:
            f.write('Item ID %s\tscore %s\n' % (iid, str(score)))
        f.write('\n')
print('3-4-3.txt completed')
```

R3-4-4



```python
# TODO: set algorithm for 3-4-4
svdpp_100 = surprise.SVDpp(n_factors=100, n_epochs=100)
svdpp_100.fit(trainset)
results = get_top_n(svdpp_100, testset, uid_list, n=10, user_based=True)

with open('3-4-4.txt', 'w') as f:
    for uid, ratings in sorted(results.items(), key=lambda x: x[0]):
        f.write('User ID %s top-10 results\n' % uid)
        for iid, score in ratings:
            f.write('Item ID %s\tscore %s\n' % (iid, str(score)))
        f.write('\n')
print('3-4-4.txt completed')
```

R3-4-5



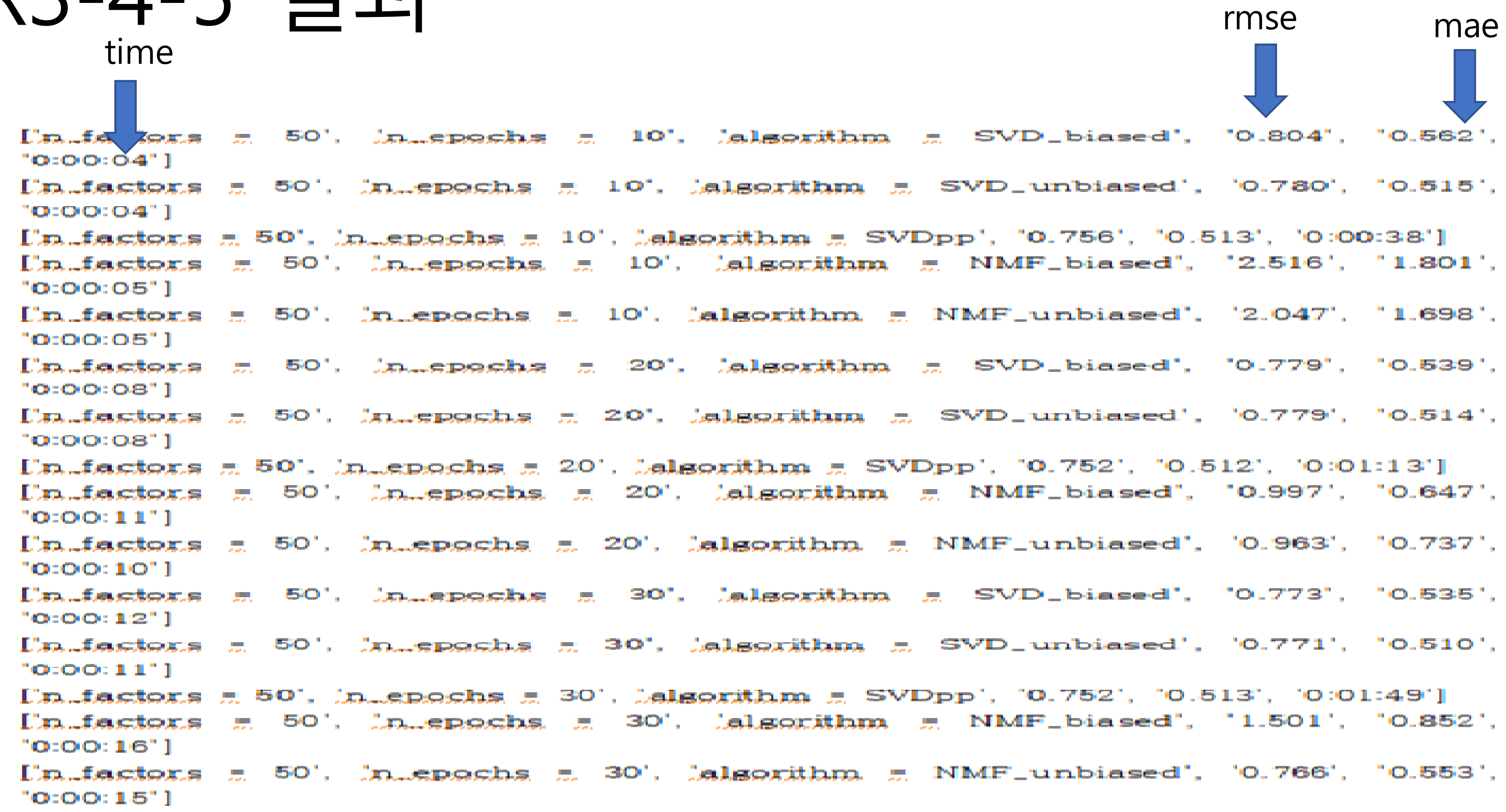
```python
# TODO: 3-4-5. Best Model
candidate_list_mf = []
for factors in range(50, 250, 50):
    for epochs in range(10, 40, 10):
        SVD_biased = surprise.SVD(n_factors=factors, n_epochs=epochs, random_state=0, biased=True)
        SVD_unbiased = surprise.SVD(n_factors=factors, n_epochs=epochs, random_state=0, biased=False)
        SVDpp = surprise.SVDpp(n_factors=factors, n_epochs=epochs, random_state=0)
        NMF_biased = surprise.NMF(n_factors=factors, n_epochs=epochs, random_state=0, biased=True)
        NMF_unbiased = surprise.NMF(n_factors=factors, n_epochs=epochs, random_state=0, biased=False)
        algo_list = [SVD_biased, SVD_unbiased, SVDpp, NMF_biased, NMF_unbiased]
        for algo in algo_list:
            start = time.time()
            cv = surprise.model_selection.cross_validate(algo, data, measures=['RMSE', 'MAE'], cv=5, verbose=False)
            cv_time = str(datetime.timedelta(seconds=int(time.time() - start)))
            mean_rmse = '{:.3f}'.format(np.mean(cv['test_rmse']))
            mean_mae = '{:.3f}'.format(np.mean(cv['test_mae']))
            if algo == algo_list[0]:
                which_algo = 'SVD_biased'
            elif algo == algo_list[1]:
                which_algo = 'SVD_unbiased'
            elif algo == algo_list[2]:
                which_algo = 'SVDpp'
            elif algo == algo_list[3]:
                which_algo = 'NMF_biased'
            else:
                which_algo = 'NMF_unbiased'
            candidate = ['n_factors = %s' % factors, 'n_epochs = %s' % epochs, 'algorithm = %s' % which_algo,
                         mean_rmse, mean_mae, cv_time]
            print(candidate)
            candidate_list_mf.append(candidate)

print('minimize mean_rmse = ', min(candidate_list_mf, key=lambda x: x[3]))
print('minimize mean_mae = ', min(candidate_list_mf, key=lambda x: x[4]))
print('minimize cv_time = ', min(candidate_list_mf, key=lambda x: x[5]))
```

R3-4-5 고려사항

3-4-5에서 고려한 사항은 다음과 같다.

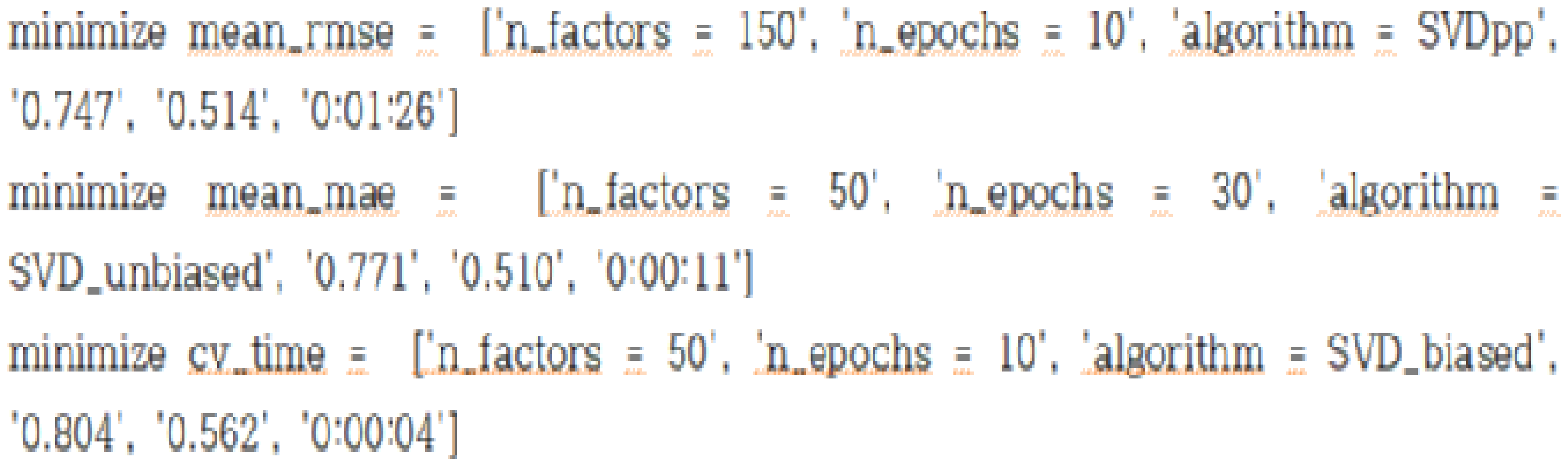
n_factors: latent factor 수 (default=100)
default 값을 중심으로 해서 50, 100, 150, 200의 값을 가지도록 설정하였다.
n_epochs: SGD 수행 횟수 (default=20)
default 값을 중심으로 해서 10, 20, 30의 값을 가지도록 설정하였다.
SVD 및 NMF의 경우에는 biased or unbiased 고려하였다.

R3-4-5 결과

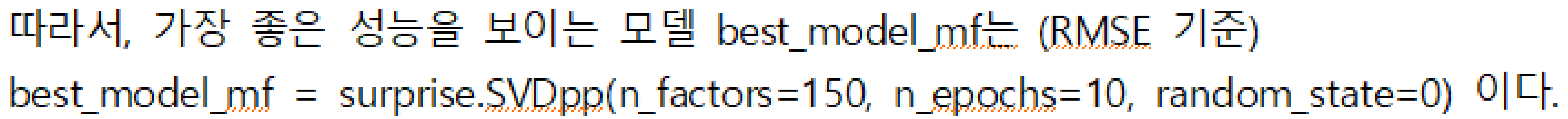


```
['n_factors = 50', 'n_epochs = 10', 'algorithm = SVD_biased', '0.804', '0.562',
'0:00:04']
['n_factors = 50', 'n_epochs = 10', 'algorithm = SVD_unbiased', '0.780', '0.515',
'0:00:04']
['n_factors = 50', 'n_epochs = 10', 'algorithm = SVDpp', '0.756', '0.513', '0:00:38']
['n_factors = 50', 'n_epochs = 10', 'algorithm = NMF_biased', '2.516', '1.801',
'0:00:05']
['n_factors = 50', 'n_epochs = 10', 'algorithm = NMF_unbiased', '2.047', '1.698',
'0:00:05']
['n_factors = 50', 'n_epochs = 20', 'algorithm = SVD_biased', '0.779', '0.539',
'0:00:08']
['n_factors = 50', 'n_epochs = 20', 'algorithm = SVD_unbiased', '0.779', '0.514',
'0:00:08']
['n_factors = 50', 'n_epochs = 20', 'algorithm = SVDpp', '0.752', '0.512', '0:01:13']
['n_factors = 50', 'n_epochs = 20', 'algorithm = NMF_biased', '0.997', '0.647',
'0:00:11']
['n_factors = 50', 'n_epochs = 20', 'algorithm = NMF_unbiased', '0.963', '0.737',
'0:00:10']
['n_factors = 50', 'n_epochs = 30', 'algorithm = SVD_biased', '0.773', '0.535',
'0:00:12']
['n_factors = 50', 'n_epochs = 30', 'algorithm = SVD_unbiased', '0.771', '0.510',
'0:00:11']
['n_factors = 50', 'n_epochs = 30', 'algorithm = SVDpp', '0.752', '0.513', '0:01:49']
['n_factors = 50', 'n_epochs = 30', 'algorithm = NMF_biased', '1.501', '0.852',
'0:00:16']
['n_factors = 50', 'n_epochs = 30', 'algorithm = NMF_unbiased', '0.766', '0.553',
'0:00:15']
```

R3-4-5 결과 (최종결과)



minimize mean_rmse = ['n_factors = 150', 'n_epochs = 10', 'algorithm = SVDpp', '0.747', '0.514', '0:01:26']
minimize mean_mae = ['n_factors = 50', 'n_epochs = 30', 'algorithm = SVD_unbiased', '0.771', '0.510', '0:00:11']
minimize cv_time = ['n_factors = 50', 'n_epochs = 10', 'algorithm = SVD_biased', '0.804', '0.562', '0:00:04']



따라서, 가장 좋은 성능을 보이는 모델 best_model_mf는 (RMSE 기준)
best_model_mf = surprise.SVDpp(n_factors=150, n_epochs=10, random_state=0) 이다.

감사합니다