데이터 마이닝 실습 중간고사

21500268, 박성찬

Task 1. 영화 분석

1. 각 변수 별로 unique한 데이터의 개수와 NA의 여부를 확인
2. 영화의 평점 점수와 개수 비교
   - 평점의 개수를 히스토그램으로 확인하여 전반적인 평점 개수의 분포를 확인
   - 평점의 평균과 개수에 대하여 산점도를 그려서 분포를 확인 및 공분산과 상관계수 확인
   - 평점의 개수 분포를 cut 함수로 나누어 label 별로의 평균 평점 확인
   - 평점의 개수가 많은 영화들이 평균 평점이 높은지 확인
3. 평점의 수가 적지만 평점이 높은 영화, 평점의 수가 많지만 평점이 낮은 영화 확인

(1) 각 변수 별로 Unique한 데이터의 개수와 NA의 여부 확인

```
## [1] "평점을 입력한 사용자의 수는 2330"
```

```
## [1] "총 영화의 코드 수는 11420"
```

```
## [1] "총 영화 수는 11016"
```

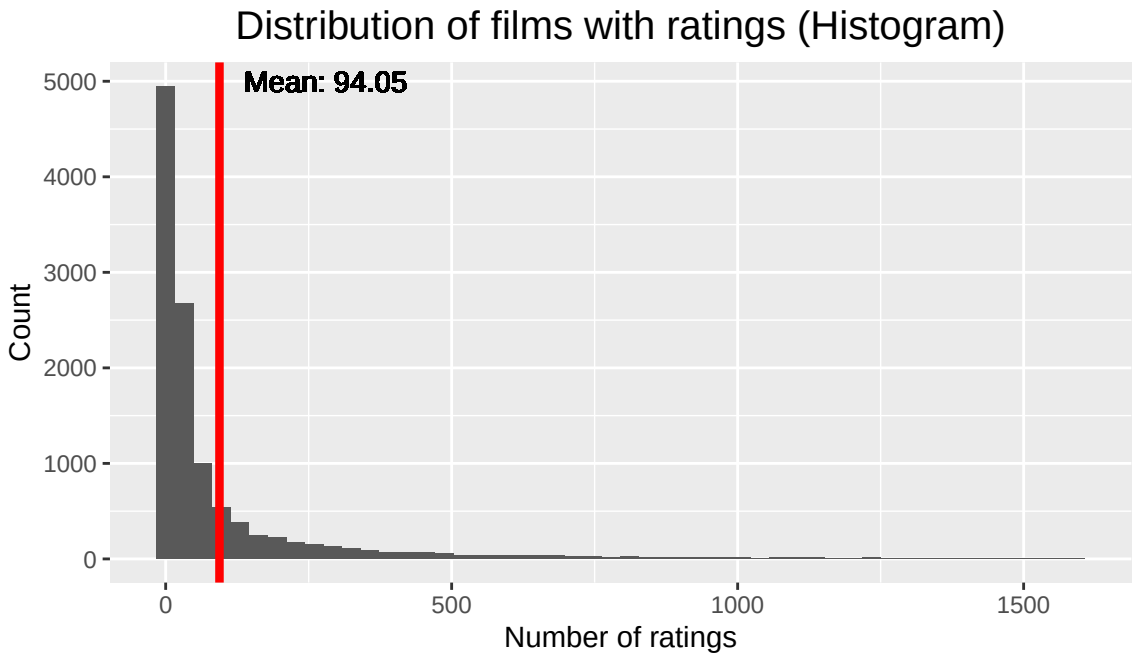
```
## [1] "NA 데이터의 개수는 0"
```

각 데이터에서 유일한 데이터의 값을 확인한다. 총 2,330명의 사용자가 평점을 입력했고, 영화의 코드 수는 11,420개, 영화 개수는 11,016개로 나오는 것을 확인했다. 영화 코드와 영화의 수가 다른 이유는 "하루, 표적, 히든" 등과 같이 다른 영화인데 영화의 이름이 동일했기 때문이었다. 또한 모든 데이터에는 NA의 개수가 없는 것을 확인했다.

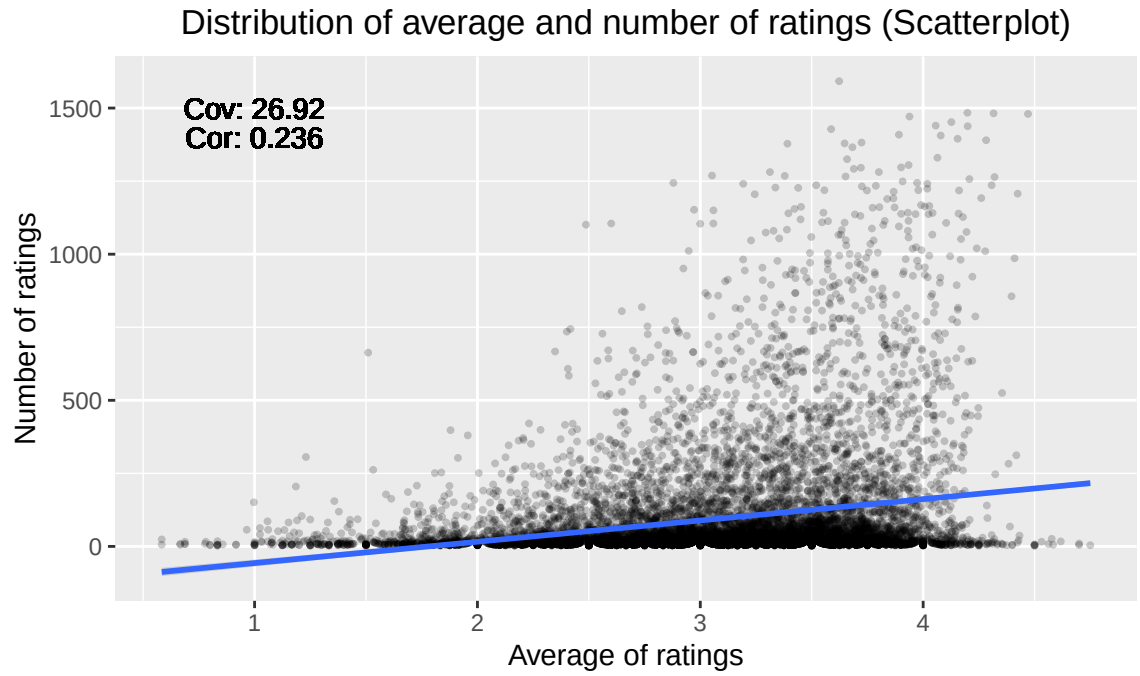
(2) 영화의 평점 점수와 개수 비교

```
## # A tibble: 3 x 6
## # Groups:   movie_code [3]
##   user_code movie_code score movie_title            score_mean     n
##   <chr>     <chr>      <dbl> <chr>                       <dbl> <int>
## 1 US000001  MC0000138    3   겨울왕국                     3.62  1592
## 2 US000001  MC0000102    2.5 인터스텔라                   4.20  1484
## 3 US000001  MC0000065    4   센과 치히로의 행방불명       4.32  1482
```

각 영화의 코드로 group_by 하여 같은 영화 별로의 평점의 점수(평균)와 개수를 비교해본다. 데이터는 위와 같음을 확인할 수 있다. 이 데이터를 통해 영화 평점 점수와 개수를 비교해보도록 한다.

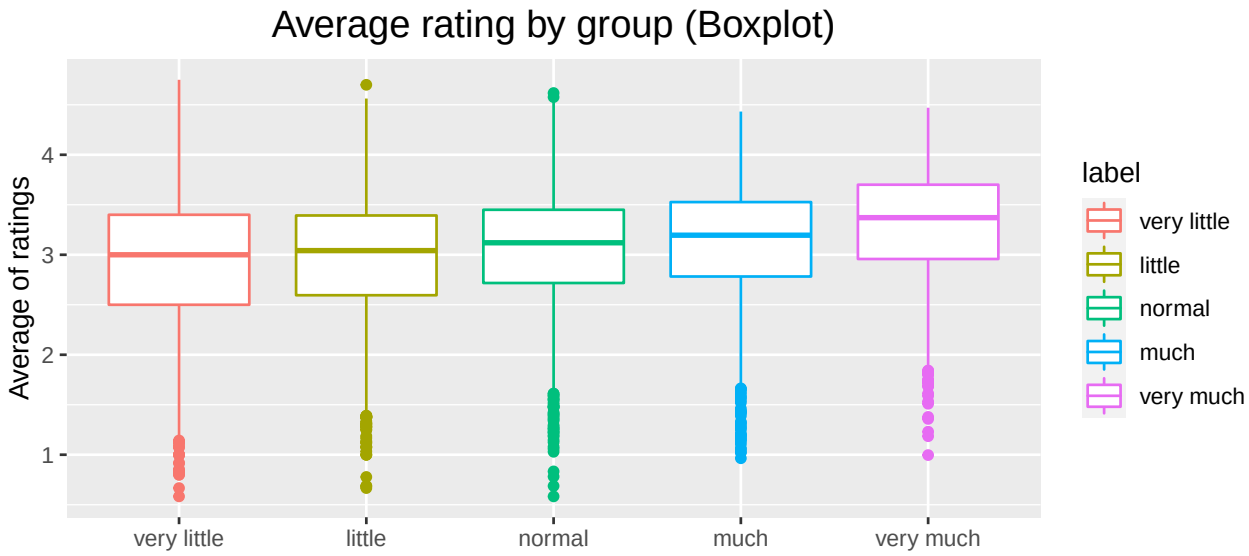


평점의 평균과 개수가 있는 데이터로부터 평점 개수에 따른 히스토그램을 통해 전반적인 분포를 확인해보도록 한다. 위 그래프에서 직관적으로 확인할 수 있듯, 대부분의 영화들의 평점의 개수는 200개 미만이라는 것을 알 수 있다.



X축을 평점의 평균으로, Y축을 평점의 개수로 하여 그래프를 확인한다. 회귀선을 확인해보면, 기울기가 낮긴 하지만 상승하는 추세에 있음을 볼 수 있다. 또한 이 두 변수의 관계를 확인해보기 위해 공분산과 상관계수를 확인해보면, 약한 양의 상관 관계가 있음을 확인할 수 있다. 이는 두 변수의 방향성이 유사함을 뜻하므로, 평점의 평균이 높을수록 평점의 개수가 높다고 이야기할 수 있다.

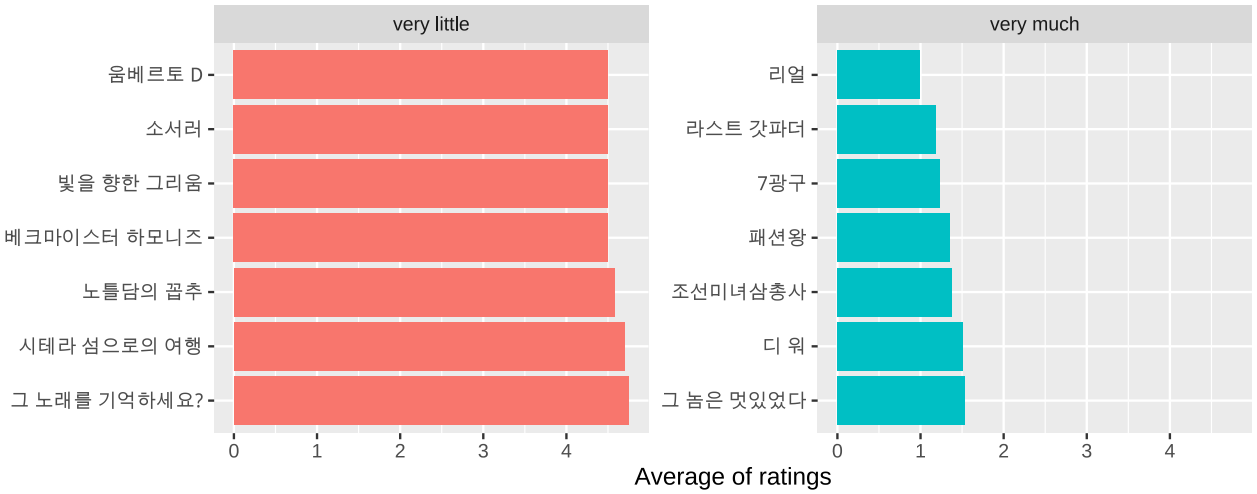
```
## # A tibble: 2 x 6
## # Groups:   movie_code [2]
##   movie_code user_code movie_title score_mean     n label
##   <chr>      <chr>     <chr>            <dbl> <int> <fct>
## 1 MC0000138  US000001  겨울왕국          3.62  1592 very much
## 2 MC0000102  US000001  인터스텔라        4.20  1484 very much
```



평점의 개수에 대한 변수에서 cut 함수를 통해 0부터 1까지 0.2 단위씩 나눠서 labeling을 해주도록 한다. 평점의 개수가 매우 많음, 많음, 보통, 적음, 매우 적음으로 각 데이터를 나눠서 평점의 개수에 따라 평점의 평균값이 어떻게 변화되는지를 확인해본다. 이를 위해서 label 별로 Boxplot을 통해 확인해보면, 평점의 개수가 많을수록 평점의 평균 값 역시 증가하는 것을 볼 수 있다. 이는 위에서 이야기했던 공분산과 상관계수 결과와도 일맥상통하는 것을 다시 한번 확인할 수 있다.

따라서 인기 있는 영화일수록 즉, 평점의 수가 많은 영화일수록 평균 평점이 높다고 이야기할 수 있다.

(3) 평점의 수가 적지만 평점이 높은 영화, 평점의 수가 많지만 평점이 낮은 영화



평점의 수가 적지만 평점이 높은 영화와 반대로 평점의 수가 많지만 평점이 낮은 영화 확인하고자 한다. 이를 위해서 앞서 만들어 놓았던 평점의 개수에 따른 label 변수를 활용한다. 이 label이 'very little'인 것은 평점의 개수가 매우 적다는 것을 의미하며 그 범위는 1개부터 7개까지이고, label이 'very much'인 것은 평점의 개수가 매우 많은 것으로 100개부터 1600개 사이의 범위이다. 이 데이터로부터 평점의 개수가 매우 적음에도 불구하고, 대부분의 사람들이 높은 평점을 매겼다면, 숨겨진 명작일 가능성이 조금이라도 높다고 생각할 수 있다. 반대로, 평점의 개수가 매우 많음에도 불구하고, 대부분의 사람들이 평점을 낮게 줬다면 호불호가 강한 영화라고 유추해볼 수 있다. 따라서 현재의 제한된 데이터 속에서 각 label 별로 평점이 가장 높았고, 낮았던 데이터 7개를 출력하면 위와 같음을 알 수 있다. 이는 test 데이터를 포함한 결과도 동일하다.

- 평점의 수가 적지만, 평점이 높은 영화: 그 노래를 기억하세요?, 시테라 섬으로의 여행, 노틀담의 꼽추
- 평점의 수가 많지만, 평점이 낮은 영화: 리얼, 라스트 갓파더, 7광구, 패션왕, 조선미녀삼총사, 디 워

Task 2. 도서 분석

1. 각 변수 별로 unique한 데이터의 개수와 NA의 여부를 확인
2. 도서의 평점 점수와 개수 비교
    - 평점의 개수를 히스토그램으로 확인하여 전반적인 평점 개수의 분포를 확인
    - 평점의 평균과 개수에 대하여 산점도를 그려서 분포를 확인 및 공분산과 상관계수 확인
    - 평점의 개수 분포를 cut 함수로 나누어 label 별로의 평균 평점 확인
    - 평점의 개수가 많은 도서들이 평균 평점이 높은지 확인
3. 평점의 수가 적지만 평점이 높은 도서, 평점의 수가 많지만 평점이 낮은 도서 확인

(1) 각 변수 별로 Unique한 데이터의 개수와 NA의 여부 확인

```
## [1] "평점을 입력한 사용자의 수는 2361"
```

```
## [1] "총 도서의 코드 수는 5865"
```

```
## [1] "총 도서 수는 5693"
```

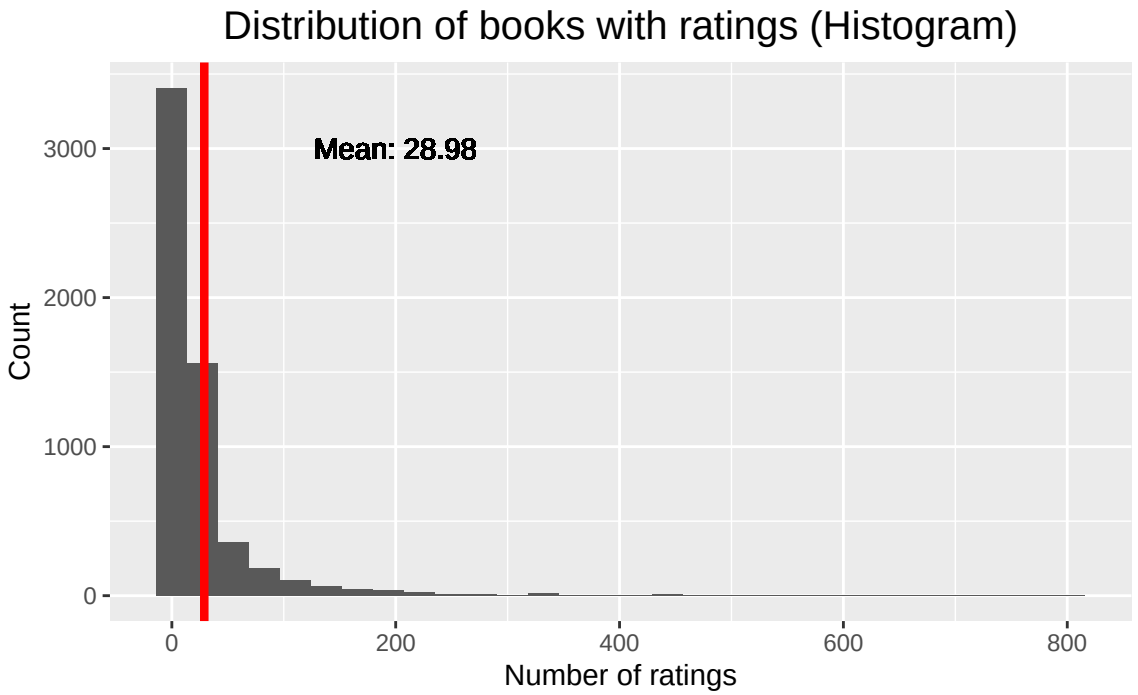
```
## [1] "NA 데이터의 개수는 0"
```

각 데이터에서 유일한 데이터의 값을 확인한다. 총 2,361명의 사용자가 평점을 입력했고, 영화의 코드 수는 5,865개, 영화 개수는 5,693개로 나오는 것을 확인했다. 영화와 동일하게 도서 역시 책의 이름이 동일한 것들이 있었기에 개수에서 약간의 차이가 있었다. 또한 모든 데이터에는 NA의 개수가 없는 것을 확인했다.

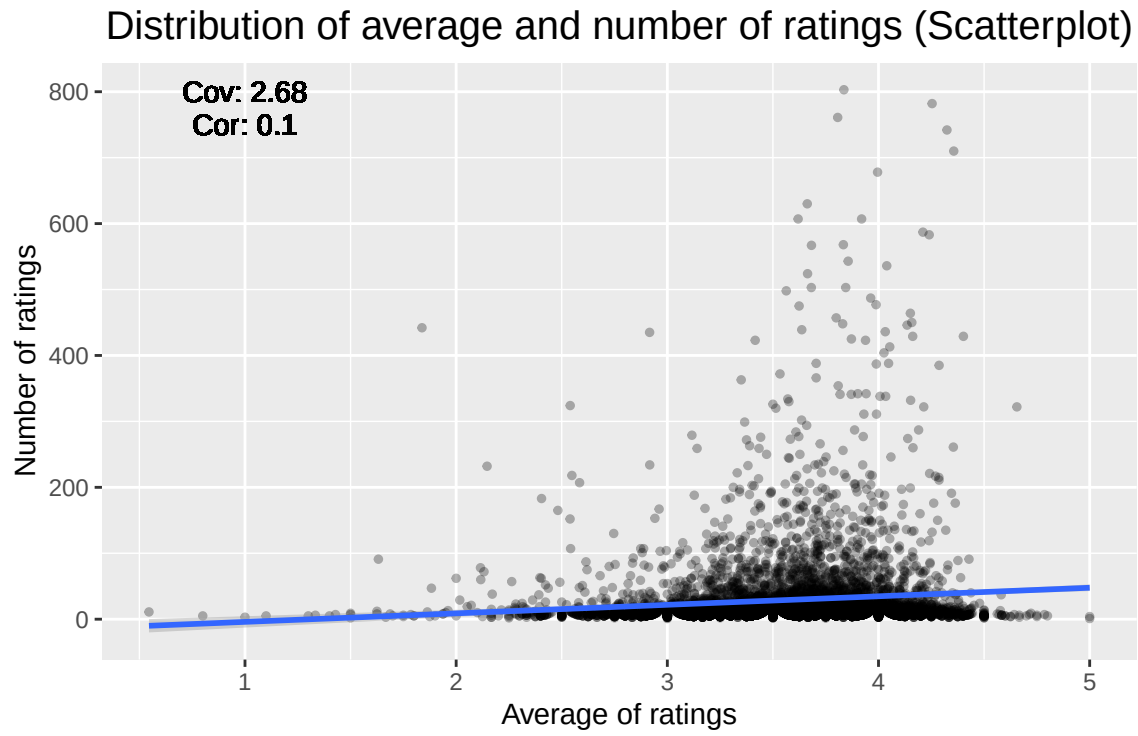
(2) 도서의 평점 점수와 개수 비교

```
## # A tibble: 3 x 6
## # Groups:   book_code [3]
##   user_code book_code score book_title           score_mean     n
##   <chr>     <chr>     <dbl> <chr>                     <dbl> <int>
## 1 US000007  BC0000691   4.5 82년생 김지영              3.84   803
## 2 US000001  BC0000051   4.5 어린 왕자                  4.25   782
## 3 US000007  BC0001348   3   나미야 잡화점의 기적       3.81   761
```

전반적인 작업 프로세스는 Task 1의 영화 분석과 동일하다. 도서 코드 별로 묶어서 평점의 평균과 개수를 넣은 데이터를 만들어준다.

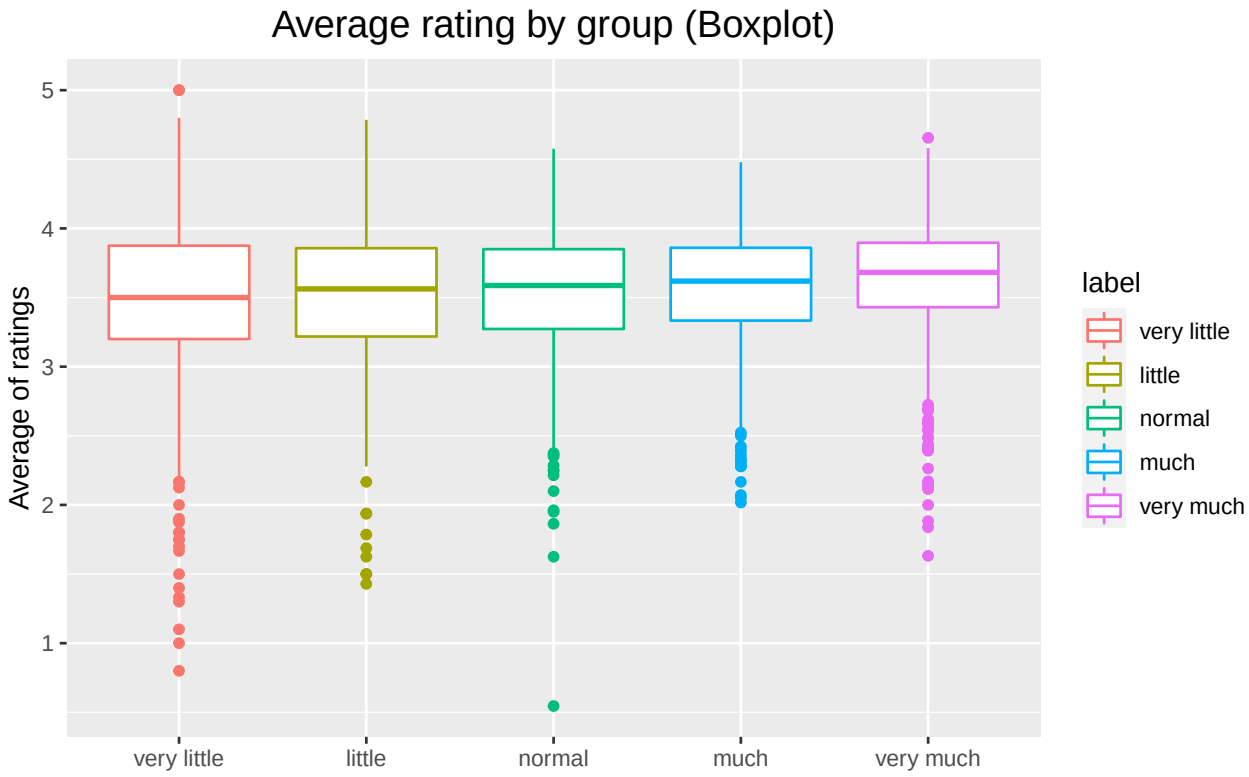


평점의 평균과 개수가 있는 데이터로부터 평점 개수에 따른 히스토그램을 통해 전반적인 분포를 확인한다. 도서 역시 영화와 유사하게, 대부분의 평점이 많이 있는 도서들이 적은 것을 확인할 수 있다.



영화에서의 과정과 유사하게 X축을 평점의 평균으로, Y축을 평점의 개수로 하여 그래프를 확인한다. 먼저 영화에서는 약 0.24 정도 됐던 상관계수가 도서에서는 0.1로 많이 떨어진 것을 확인할 수 있다. 상관계수는 -1부터 1까지의 범위에 속하는데, -1이라면 음의 상관 관계로 두 변수가 완전히 반대로 움직이는 것이고, 1이라면 양의 상관 관계로 같은 방향으로 움직이는 것이다. 도서의 경우에는 상관계수가 0.1이므로 거의 0에 가까운데, 이는 두 변수 사이에 관계를 정의하기 어려운 상황이라고 이야기할 수 있다. (상관 관계가 없다.) 한편, 회귀선의 기울기를 보면, 미세하게나마 증가하는 추세를 보이긴 한다.

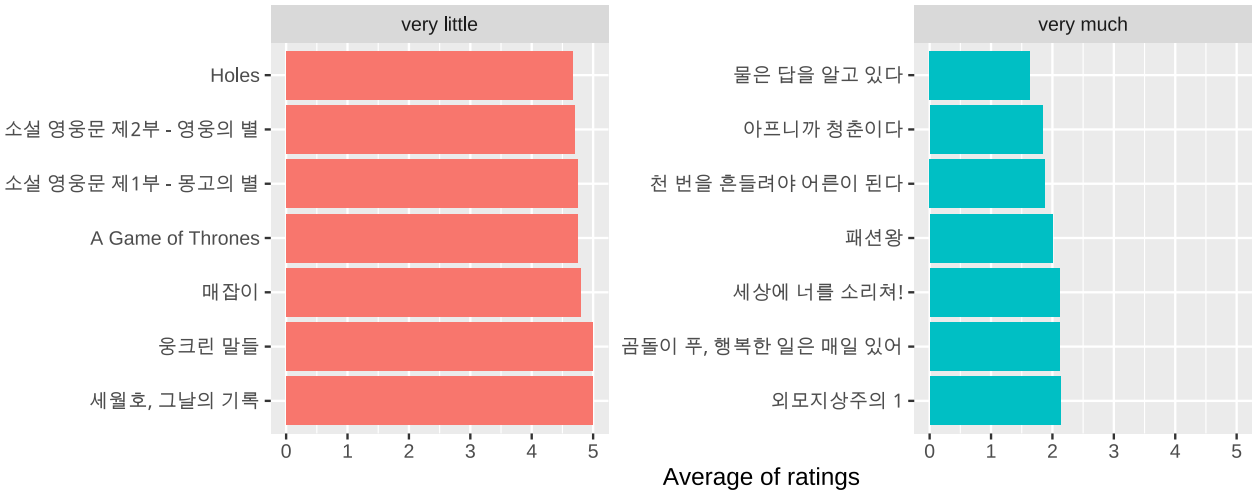
```
## # A tibble: 2 x 6
## # Groups:   book_code [2]
##   book_code user_code book_title    score_mean     n label
##   <chr>     <chr>     <chr>              <dbl> <int> <fct>
## 1 BC0000691 US000007  82년생 김지영       3.84   803 very much
## 2 BC0000051 US000001  어린 왕자           4.25   782 very much
```



영화와 동일하게 평점의 개수에 따라 label을 나누고, Boxplot을 통해 전반적인 분포를 확인해본다. 영화 보다는 그 차이가 확연히 드러나지는 않지만, 그럼에도 불구하고 도서 역시도 평점의 개수가 많을수록 평점의 평균이 상승하는 것을 미세하게나마 확인할 수 있다. 산점도에서 볼 수 있듯, 회귀선 자체도 어느 정도는 우상향 하고 있으며, 데이터의 분포 역시 평점의 개수가 많을수록 평점의 평균이 높은 것을 육안으로 확인할 수 있다. 물론 공분산이나 상관계수 등과 같이 수치적으로는 앞서 확인했듯, 그렇게 높은 편은 아니긴 하지만 어느 정도는 많은 평점의 수를 가진 도서들이 평점의 평균도 높다고 할 수 있다.

따라서 영화보다는 그 관계가 확연히 드러나지는 않지만, 도서 역시도 어느 정도는 인기 있는 도서 즉, 평점의 수가 많은 도서 일수록 평균 평점이 조금 더 높다고 이야기할 수 있다.

(3) 평점의 수가 적지만 평점이 높은 도서, 평점의 수가 많지만 평점이 낮은 도서



평점의 수가 적지만 평점이 높은 영화와 반대로 평점의 수가 많지만 평점이 낮은 영화 확인하고자 한다. 사실 주어진 데이터가 영화와 도서 종류, 평점 그리고 인원 수 정도 밖에 없기 때문에 다음과 같은 분석을 하기에는 비교적 제한적이라고 이야기할 수 있다. 하지만 현재 있는 데이터로써 잘 알려지지 않은 명작을 찾거나, 호불호가 갈리는 영화나 도서를 확인하면 다음과 같다. 도서의 결과 역시 영화의 결과와 동일한 방식으로 구해보았다. 결과를 확인해보니, 영화에서는 흥행하지 못했다는 평이 있었던 '리얼'과 같은 영화들이 나옴으로써 어느 정도 그 결과가 타당하다고 생각했는데, 도서는 조금 의외의 결과가 나왔다. 김난도 교수의 '아프니까 청춘이다'나 '천 번을 흔들려야 어른이 된다', '곰돌이 푸, 행복한 일은 매일 있어' 등의 책들은 베스트셀러 구역에서 자주 봤던 책들이었는데 비교적 낮은 평점을 받았던 것을 확인할 수 있었다.

- 평점의 수가 적지만, 평점이 높은 도서: 세월호 그날의 기록, 웅크린 말들, 매잡이, A Game of Thrones
- 평점의 수가 많지만, 평점이 낮은 도서: 물은 답을 알고 있다, 아프니까 청춘이다, 패션왕

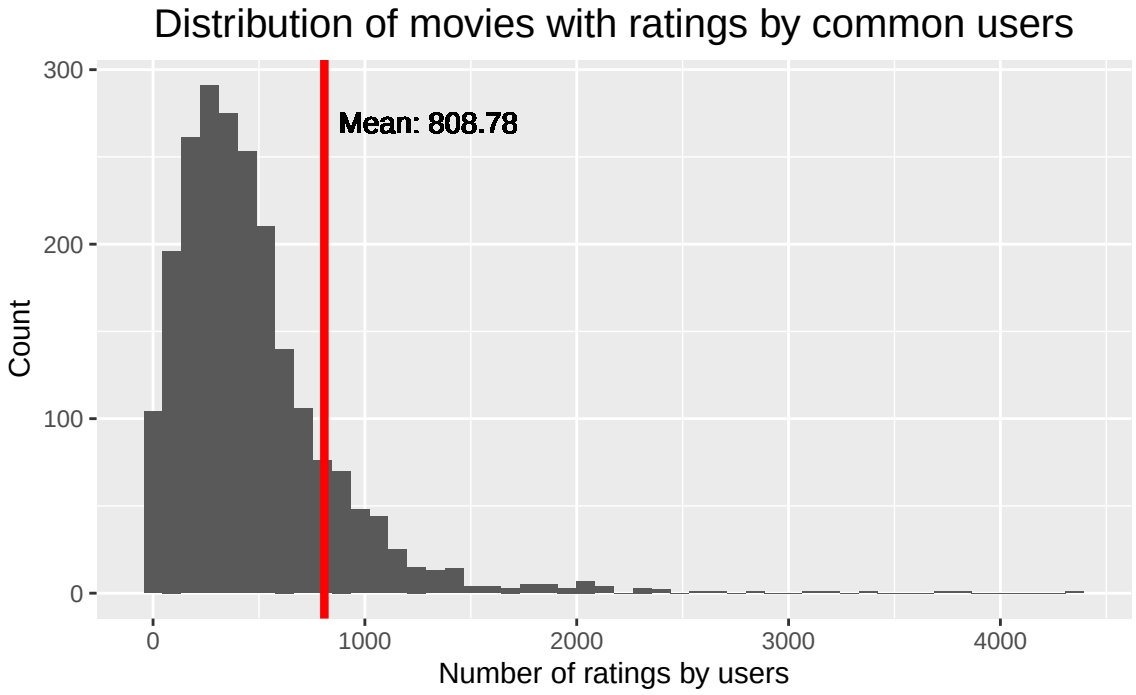
Task 3. 사용자 분석

1. 도서와 영화 모두에서 평점 기록이 있는 사용자 추리기
   - 원그래프를 통해서 도서와 영화 모두에서 평점 기록이 있는 사용자의 비율 확인
2. 영화 평점 기록이 많은 사용자들이 도서 평점 기록도 많은지 확인
   - 영화 평점 기록에 따라 히스토그램으로 전반적인 분포 확인
   - 영화 평점 기록을 기준으로 영화와 도서 모두에 labeling 하여 그 관계성이 유사한지 확인
3. 사용자의 평점을 매기는 기준이 영화와 도서에서 동일한지 확인
   - 같은 사용자에 대하여 X축을 영화의 평점, Y축을 도서의 평점으로 그래프 표현하여 확인
   - 영화 평점이 후한 사용자는 도서 평점도 후한지 확인

(1) 도서와 영화 모두에 평점 기록이 있는 사용자 추출

도서와 영화 모두에서 `user_code`를 변수를 이용해 평점 기록이 모두 있는 사용자를 확인해보도록 한다. `intersect` 함수를 통해서 도서와 영화의 교집합 사용자 코드를 확인하면 총 2,190명의 사용자가 나오는 것을 확인했다. 이를 직관적으로 확인하기 위해 전체 사용자들 대비 평점 기록이 모두 있는 사용자 (`common`)와 둘 중 하나만 있는 사용자 (`rare`)를 확인해보니 약 95% 정도가 공통된 사용자임을 확인할 수 있었다.

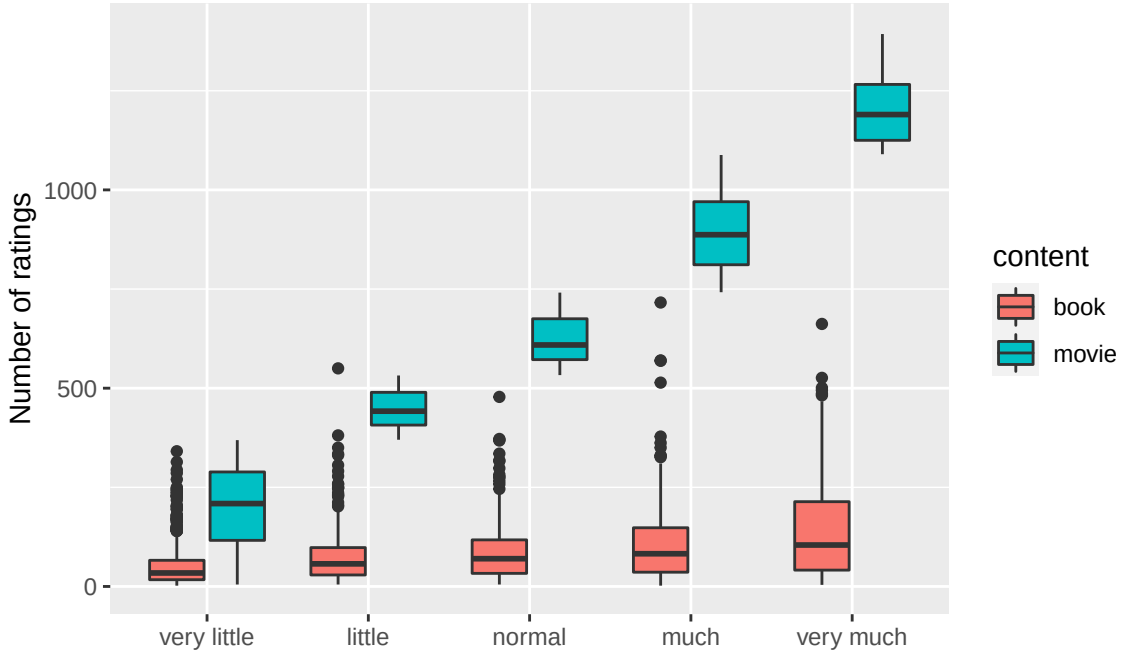
(2) 영화 평점 기록이 많은 사용자들이 도서 평점 기록도 많은지 확인



영화의 평점을 많이 기록한 사람들이 도서의 평점도 많이 기록했는지 확인해보려고 한다. 이를 위해서 가장 먼저, 영화와 도서 모두 평점을 기록한 사람들에 대하여 사용자 별로 영화의 평점을 몇 개 정도 입력했는지를 확인할 수 있는 히스토그램을 그려본다. X축은 각 사용자 별로 평점을 기록한 숫자를 의미하고, Y축은 각 범위 별로 몇 명 정도 사용자가 분포되었는지를 의미한다. 그 평균값은 약 808개 정도이며, 전반적으로는 1,000개 이하로 평점을 작성하고 있음을 직관적으로 확인할 수 있다.

```
##   0%  20%  40%  60%  80% 100%
##    5  369  532  741 1088 4357
```

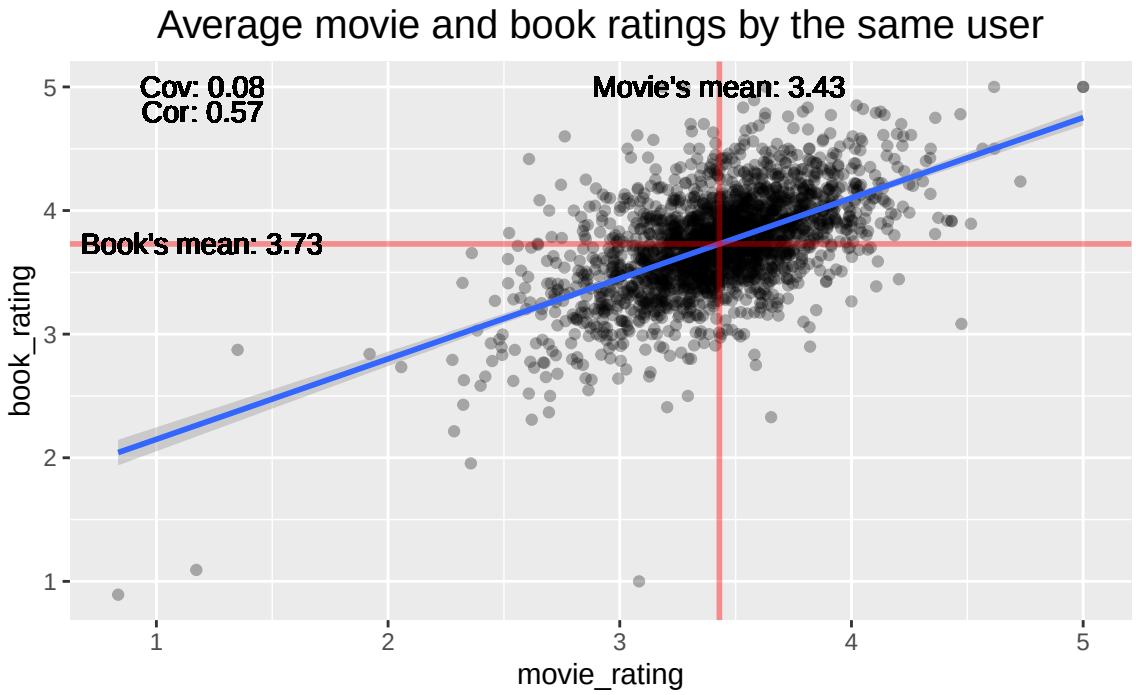
```
##         label     movie      book
## 1 very little  254.3220  93.63306
## 2      little  451.9074 132.73703
## 3      normal  628.0936 151.84116
## 4        much  902.9572 204.97112
## 5   very much 1810.8021 282.11733
```



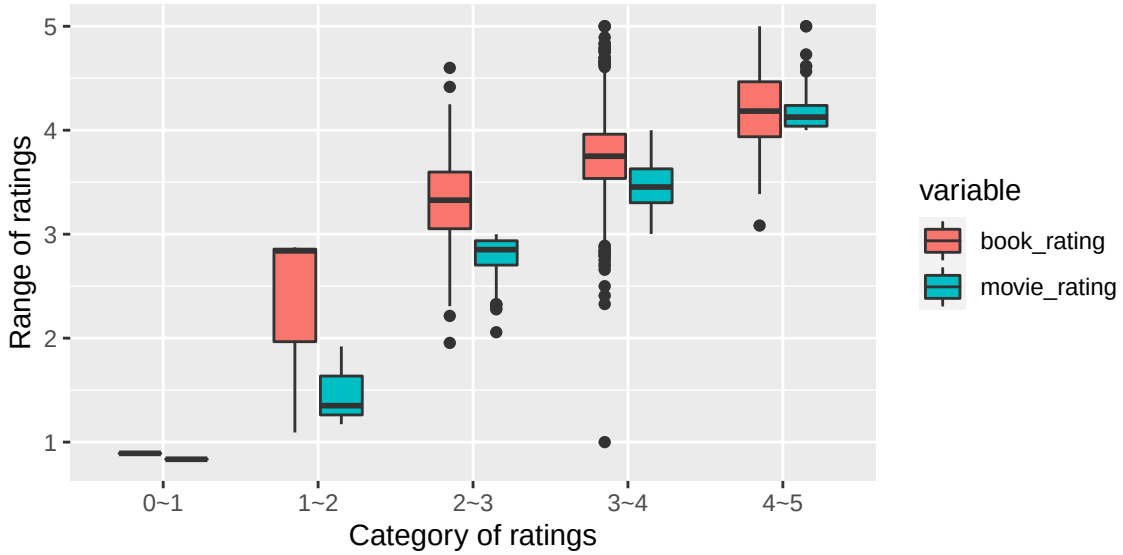
영화의 평점 기록이 많은 사용자를 구분하기 위해 quantile 함수를 통해 20%씩 나눠서 총 5개의 label을 만들어본다. 이는 한 사람이 약 1,088개 이상의 평점을 기록했다면 영화를 많이 보면서 평점을 많이 기록한 사용자로 정의할 수 있다. 이번에는 영화를 많이 보는 사람들이 도서에 대해서도 동일한지 확인하는 것이므로, 'very little', 'little', 'normal', 'much', 'very much'로 나뉜 사용자 명단을 그대로 도서 데이터로 가져와서 각각의 값을 비교해본다. 영화의 평점 개수로 label을 나눴다보니, 영화를 의미하는 파란색 박스는 오른쪽으로 가면서 점차적으로 증가하는 것을 확인할 수 있다. 그렇다면 이 관계가 빨간색 박스인 도서에도 동일할까? 일단 박스플랏 결과로만 봐서는 어느 정도는 비슷한 움직임을 보인다고 할 수 있다. 이를 label 별로 영화와 도서의 평균으로 계산해보면, 영화의 평점 개수가 많은 사용자일수록 도서의 평점 개수도 증가하는 양의 상관관계를 보이고 있다.

따라서 비록 영화만큼 급격히 증가하지는 않지만, 영화를 많이 보는 사용자들이 도서에 대한 평점 기록 역시 비교적 많다고 이야기할 수 있다.

(3) 사용자의 평점을 매기는 기준이 영화와 도서에서 동일한지 확인



이번에는 영화와 도서에서 평점을 매기는 기준이 비슷한지 확인해보려고 한다. 이를 위해서 같은 사용자에 대하여 영화와 도서 각각으로 평점의 평균을 계산하여 X축은 영화의 평점 평균을, Y축은 도서의 평점 평균으로 하여 산점도를 그려보았다. 이를 통해 전반적인 모양이 Y = X 형태의 우상향 그래프를 보인다면, 영화와 도서 모두 같은 점수를 부여했다고 이해할 수 있다. 실제 그래프를 확인해보니, 전반적으로 회귀선의 기울기 역시 우상향하는 모양을 그리는 것을 볼 수 있었고, 상관계수 역시 0.57로 매우 높은 상관관계를 보이고 있음을 확인했다. 한가지 주목할 것은 전반적으로 영화와 도서 모두 3~4점 대의 평점이 매우 모여 있는 것을 확인할 수 있다.



따라서 전반적으로 영화와 도서에서 평점을 매기는 기준이 거의 유사하며, 박스플랏을 통해 영화 평점이 generous한 사용자는 도서 평점도 generous 하고, 그 반대도 그러하다고 이야기할 수 있다.

Task 4. 영화 추천

1. recommenderlab 패키지에서 사용되는 추천 알고리즘 소개 (recommenderRegistry) 및 전처리
   - UBCF_realRatingMatrix, IBCF_realRatingMatrix 방식과 parameter 확인
   - train과 test 데이터에서 공통된 영화만 추리는 전처리 수행
2. 영화에 대해 train 데이터를 바탕으로 UBCF, IBCF 모델로 test 데이터를 예측
   - 이웃의 개수를 의미하는 파라미터를 수정하며 MAE가 최소가 되는 최적 모델 설정
   - train 데이터로부터 가장 MAE가 낮은 모델을 test에 적용하여 결과 확인 및 비교

(1) recommenderlab 패키지에서 사용되는 추천 알고리즘 소개 및 전처리

```
## $UBCF_realRatingMatrix
## Recommender method: UBCF for realRatingMatrix
## Description: Recommender based on user-based collaborative filtering.
## Reference: NA
## Parameters:
##     method nn sample weighted normalize min_matching_items min_predictive_items
## 1 "cosine" 25  FALSE     TRUE  "center"                  0                   0

## $IBCF_realRatingMatrix
## Recommender method: IBCF for realRatingMatrix
## Description: Recommender based on item-based collaborative filtering.
## Reference: NA
## Parameters:
##    k   method normalize normalize_sim_matrix alpha na_as_zero
## 1 30 "Cosine"  "center"                FALSE   0.5      FALSE
```

실제 추천 모델을 만들어보는 이번 문제에서는 UBCF와 IBCF 모델을 train 데이터에 학습하여 test 데이터의 고객 평점을 예측해보도록 한다. recommenderRegistry 함수를 통해서 패키지에서 사용되는 함수들의 전반적인 설명들을 확인할 수 있는데, 이번에는 UBCF와 IBCF 모델을 사용하므로 이 두 가지 내용을 확인한다. UBCF 알고리즘은 그 이름처럼 사용자를 중심으로 추천 모델을 만드는 것이고, IBCF는 item 즉, 이 문제에는 영화를 중심으로 추천 모델을 만드는 것이다. 사용하는 Recommender 함수의 parameter 중에서 method에 UBCF와 IBCF를 입력하여 모델을 만들 수 있다. 그 전에 패키지 자체에 기본적으로 내장되어 있는 default parameter는 다음과 같다.

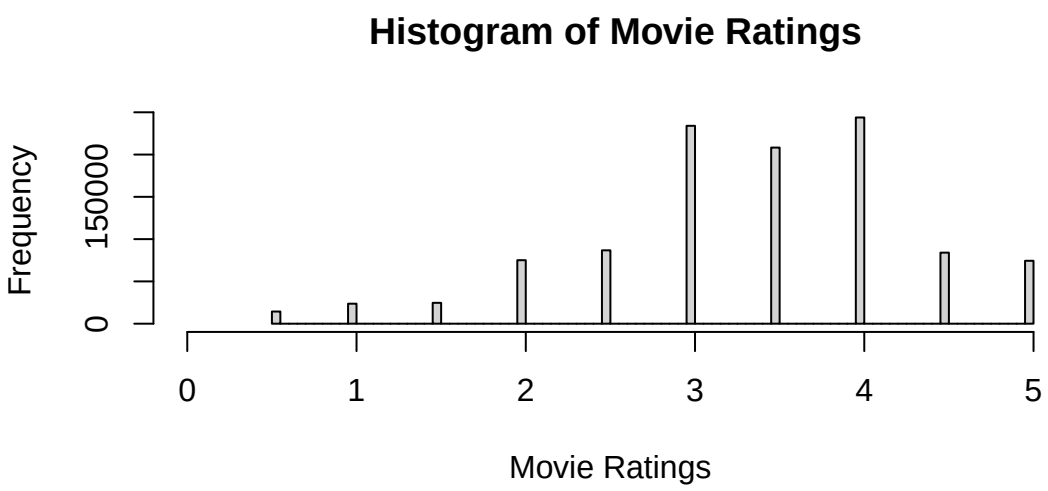
- UBCF: 'center' 방식 normalize, cosine 유사도로 유사한 25명의 사용자(nn)로부터 rating
- IBCF: 'center' 방식 normalize, cosine 유사도로 유사한 30개의 item(k)으로부터 rating

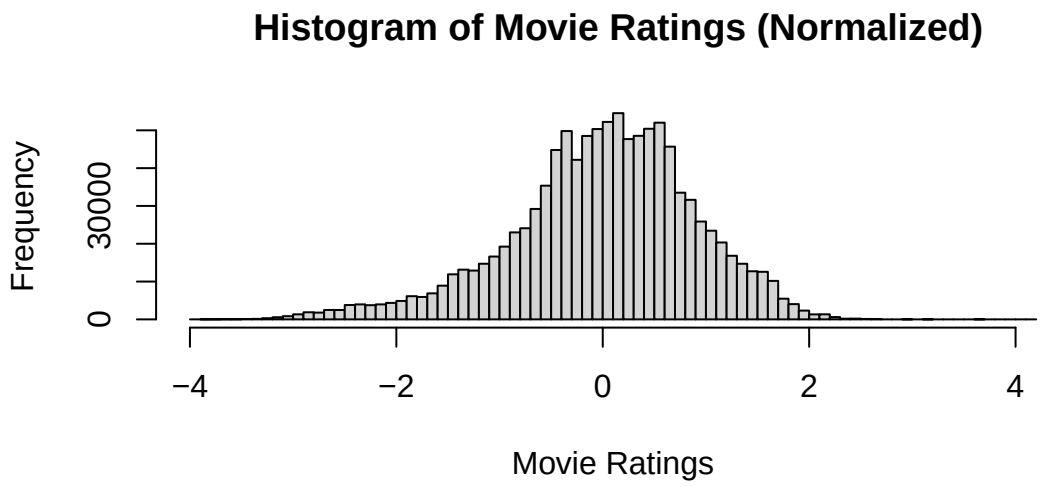
```
## [1] "train data에서의 유일한 movie_code 개수: 11420"

## [1] "test data에서의 유일한 movie_code 개수: 10815"

## [1] "train과 test 데이터의 교집합 개수: 10815"
```

이 문제에서 추천 모델의 목적은 train 데이터를 통해서 특정 사용자와 비슷한 사용자군의 데이터를 통해 test 데이터에 존재하는 영화의 평점을 예측하고, 실제 평점과 예측 평점을 통해 MAE를 계산하는 것이다. 따라서 바로 추천 모델을 만들기 전에 target이 될 수 있는 영화 즉, test 데이터에 존재하는 movie_code를 train 데이터에 맞춰주는 작업을 수행하도록 한다. 확인해보니, train 데이터에서 유일한 movie_code의 개수는 11,420개 이고, test 데이터에서는 10,815개, 그리고 intersect 함수를 통한 교집합은 test와 동일한 10,815개이므로, train 데이터에서 공통된 movie_code 기준으로 filter를 하도록 한다.

```
## 2330 x 10815 rating matrix of class 'realRatingMatrix' with 1069103 ratings.

## 2330 x 10815 rating matrix of class 'realRatingMatrix' with 268512 ratings.

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     5.0   206.0   378.0   458.8   588.0  4240.0

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1.00    9.00   25.00   98.85   86.00 1592.00
```

그 후에는 영화 코드와 평점을 spread 하여 row는 사용자가, column에는 영화가 들어가고 각 셀에는 해당 사용자가 각 영화에 대해 입력한 평점이 들어가 있는 matrix를 만들고, 추천 시스템에 활용될 수 있도록 realRatingMatrix 데이터 구조로 변환해주도록 한다. 그렇게 한 후의 결과를 보면, train과 test 데이터에서 총 2,330명의 사용자와 10,815개의 영화에 대한 평점이 담긴 데이터를 확인할 수 있다. 그리고 train 데이터에 대하여 rowCounts와 colCounts의 summary를 통해 사용자 중에서 가장 많이 영화를 본 사람은 총 4,240 편의 영화를 봤고, 영화 중에서 총 1,592명의 사용자가 평점을 입력한 영화가 가장 사용자에게 인기 있었다고 할 수 있다.





또한 train 데이터에 대해 영화의 평점을 바탕으로 정규화 여부에 따른 히스토그램을 그려보았고, 전반적으로 3~4점 사이의 평점이 가장 많았던 것을 볼 수 있다. 이러한 내용을 바탕으로 실제 평점을 예측할 수 있는 UBCF, IBCF 모델을 각각 만들고, 평가 지표인 MAE를 비교해본다.

(2) 영화에 대해 train 데이터를 바탕으로 UBCF, IBCF 모델로 test 데이터를 예측

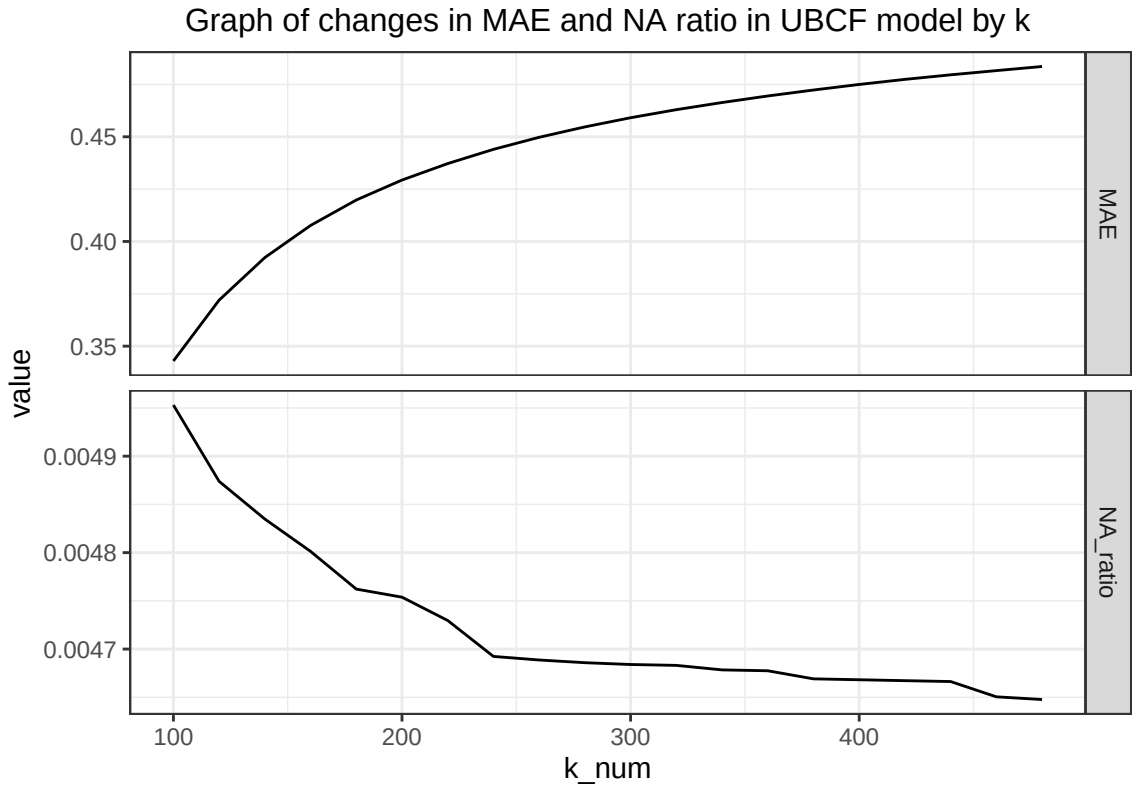
```
##### UBCF model
UBCF_model <- Recommender(movie_rrm, method = 'UBCF')

train_pred <- predict(UBCF_model, movie_rrm, type = 'ratingMatrix')
train_pred <- as(train_pred, 'data.frame')
colnames(train_pred) <- c('user_code', 'movie_code', 'score_pred')

UBCF_movie <- movie_test %>%
  left_join(train_pred, by = c('user_code', 'movie_code')) %>%
  as.data.frame()

UBCF_movie %>%
  summarise(MAE = mean(abs(score - score_pred), na.rm = T))
```

```
##### IBCF model
IBCF_model <- Recommender(movie_rrm, method = 'IBCF')

train_pred <- predict(IBCF_model, movie_rrm, type = 'ratingMatrix')
train_pred <- as(train_pred, 'data.frame')
colnames(train_pred) <- c('user_code', 'movie_code', 'score_pred')

IBCF_movie <- movie_test %>%
  left_join(train_pred, by = c('user_code', 'movie_code')) %>%
  as.data.frame()

IBCF_movie %>%
  summarise(MAE = mean(abs(score - score_pred), na.rm = T))
```

```
##            MAE  NA_ratio
## UBCF 0.6970639 0.7759877
## IBCF 0.8511602 0.8019828
```

주어진 train 데이터를 바탕으로 recommenderlab 패키지를 활용하여 평점을 예측할 수 있는 추천 모델을 만들도록 한다. 이번 문제에서는 협업 필터링 모델의 대표적인 알고리즘인 UBCF와 IBCF를 활용하도록 한다. 기본적인 파라미터는 수정하지 않고, 앞서 이야기했던 것처럼 모델이 default로 가지고 있는 코사인 유사도를 바탕으로 각각 nn과 k가 25, 30 일 때 결과를 확인하도록 한다. train 데이터에 대하여 위와 같이 각각 모델들을 학습시키고, ratingMatrix를 통해서 최종적인 평점까지 출력하도록 한다. 그 후에는 사용자의 코드와 영화 코드가 같은 것들을 기준으로 test 데이터와 예측한 데이터를 join 하여 실제와 예측값을 하나의 데이터프레임으로 만들어주도록 한다. 최종적으로 각 데이터프레임에서 MAE를 구하기 위하여 mean(abs(score - score_pred))를 계산해주고, 최종적인 결과를 출력하면 다음과 같음을 확인할 수 있다. 물론 이 모델들이 가지고 있는 한계점도 분명히 있다. 출력된 결과를 보면 MAE와 함께 각 데이터에서 NA이 비율까지 출력해보았는데, 전반적으로 많은 데이터들이 NA를 가지고 있음을 확인할 수 있다. 즉, 많은 데이터를 가지고 있는 것에 비해 충분한 이웃의 수를 고려하지 못해서 예측에 어려움이 있었던 것으로 생각된다. 따라서 이웃의 개수를 의미하는 파라미터를 조정하여 조금 더 개선된 결과를 얻을 수 있을지 다시 확인해보도록 한다.

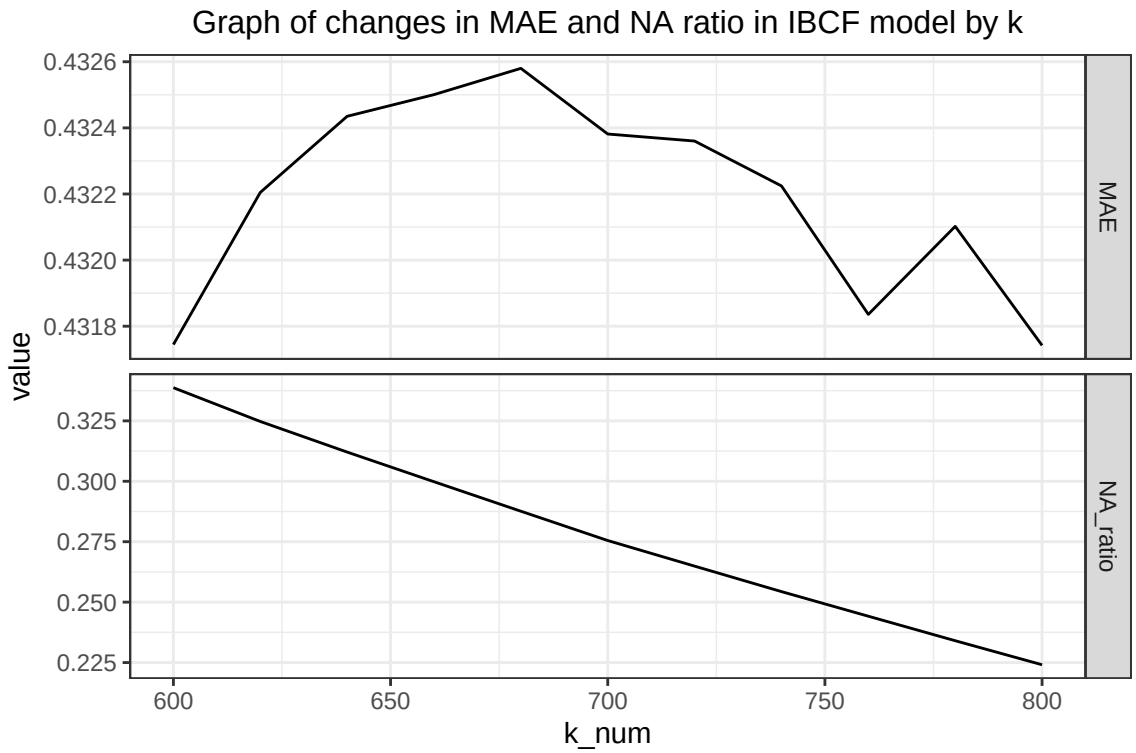
```r
# UBCF model - changing neighbor parameter
result_df <- data_frame()
for (k in seq(100, 500, by = 20)){
  UBCF_model <- Recommender(movie_rrm, method = 'UBCF', parameter = list(nn = k))

  train_pred <- predict(UBCF_model, movie_rrm, type = 'ratingMatrix')
  train_pred <- as(train_pred, 'data.frame')
  colnames(train_pred) <- c('user_code', 'movie_code', 'score_pred')

  UBCF_result_test <- movie_test %>%
    left_join(train_pred, by = c('user_code', 'movie_code')) %>%
    as.data.frame()
  UBCF_result_train <- movie_train %>%
    left_join(train_pred, by = c('user_code', 'movie_code')) %>%
    as.data.frame()

  train <- UBCF_result_train %>%
    summarise(MAE = mean(abs(score - score_pred), na.rm = T),
              NA_ratio = mean(is.na(score_pred))) %>%
    mutate(k_num = k, df = 'train', model = 'UBCF', sim = 'cosine', norm = 'center')
  test <- UBCF_result_test %>%
    summarise(MAE = mean(abs(score - score_pred), na.rm = T),
              NA_ratio = mean(is.na(score_pred))) %>%
    mutate(k_num = k, df = 'train', model = 'UBCF', sim = 'cosine', norm = 'center')

  train_test <- bind_rows(train, test)
  result_df <- bind_rows(result_df, train_test)
}
```



seq 함수를 이용하여 100부터 500까지 20씩 증가하는 값을 UBCF 모델에서 이웃의 개수를 의미하는 nn 변수에 할당하여 for loop을 수행하도록 한다. 이를 통해 nn의 값이 변화하면서 만들어진 모델이 예측하는 평점들을 데이터프레임으로 만들고 train과 test 데이터에 대하여 join 해서 각각 모델의 성능인 MAE와 NA ratio를 계산해보도록 한다. 그 결과를 시각화해서 보면 위와 같음을 볼 수 있다. 이웃의 개수를 확인할수록 더 많은 데이터를 확인하다보니, 전반적으로 예측하지 못하는 즉, NA를 반환하는 값들은 계속 낮아지고 있는 것을 볼 수 있다. 하지만 그에 따라 MAE는 계속해서 상승하고 있음도 확인할 수 있다. 그나마 이 결과에서 가장 최적의 모델을 선택하면, NA ratio가 감소하는 그 감소폭이 유난히 적은 k가 약 240 정도일때가 최적이라고 이야기할 수 있을 것 같다. 따라서 k가 240 일때 test 데이터에서의 성능은 어떻게 나왔는지 밑의 IBCF 모델의 결과와 함께 아래서 확인해보도록 한다.

```
# IBCF model - changing neighbor parameter
result_df <- data_frame()
for (k in seq(600, 800, by = 20)){
  UBCF_model <- Recommender(movie_rrm, method = 'IBCF', parameter = list(k = k))

  train_pred <- predict(UBCF_model, movie_rrm, type = 'ratingMatrix')
  train_pred <- as(train_pred, 'data.frame')
  colnames(train_pred) <- c('user_code', 'movie_code', 'score_pred')

  UBCF_result_test <- movie_test %>%
    left_join(train_pred, by = c('user_code', 'movie_code')) %>%
    as.data.frame()
  UBCF_result_train <- movie_train %>%
    left_join(train_pred, by = c('user_code', 'movie_code')) %>%
    as.data.frame()

  train <- UBCF_result_train %>%
    summarise(MAE = mean(abs(score - score_pred), na.rm = T),
              NA_ratio = mean(is.na(score_pred))) %>%
    mutate(k_num = k, df = 'train', model = 'IBCF', sim = 'cosine', norm = 'center')
  test <- UBCF_result_test %>%
    summarise(MAE = mean(abs(score - score_pred), na.rm = T),
              NA_ratio = mean(is.na(score_pred))) %>%
    mutate(k_num = k, df = 'test', model = 'IBCF', sim = 'cosine', norm = 'center')

  train_test <- bind_rows(train, test)
  result_df <- bind_rows(result_df, train_test)
}
```

동일한 작업을 이번에는 IBCF 모델에도 적용해보고, 그 결과를 비교해본다. 앞선 UBCF 코드 과정과 거의 유사하고, 추천 시스템의 method와 이웃의 개수를 의미하는 k 값을 바꿔주면서 결과가 어떻게 나오는지 확인해보도록 한다. IBCF 모델은 그 이름처럼 item 기반으로 결과를 만들어내는 모델이므로, UBCF 모델과 달리 k의 범위를 조금 크게 600부터 800으로 잡고, 모델을 만들어보았다. 최종적으로 IBCF 모델의 k 값의 변화에 따른 MAE와 NA ratio 값은 다음과 같음을 확인할 수 있다.



IBCF 모델 역시 k의 값이 증가하면서 NA의 개수가 줄어드는 것을 확인할 수 있고, MAE의 값도 변화하고 있는 것을 확인할 수 있다. 이 모델을 통한 결과로는 k가 약 760 정도일 경우에 MAE가 소폭 감소하는 모양을 보이므로 그때가 최적의 모델이라고 이야기할 수 있겠다. 따라서 이 때 test 데이터에 대한 결과를 UBCF 모델과 함께 비교해보도록 한다.

```
##       k_num       MAE   NA_ratio
## UBCF    240 0.5648040 0.02384623
## IBCF    760 0.7159554 0.17935511
```

최종적으로 UBCF와 IBCF로 만든 모델들의 성능을 비교해보면, 파라미터의 기본값을 사용했을 경우에 NA의 개수가 많긴 했지만, train 데이터에서 UBCF는 MAE가 0.69이고, IBCF는 0.85라는 것을 알 수 있다. 또한 NA를 조금이나마 줄이기 위해 시도했던 k 값의 변화에 따른 모델의 성능을 확인하더라도 UBCF 모델이 NA 비율이 현저히 낮으면서 모델이 갖는 성능 지표인 MAE도 낮았다. 뿐 만 아니라, train 데이터를 통한 최적 모델에 대하여 test 데이터에서 MAE 값을 확인해보면, 일반적으로 알려진 것처럼 IBCF 모델(0.71)보다 UBCF 모델(0.56)의 성능이 더 좋다고 이야기할 수 있겠다.

Task 5. 도서 추천

1. 도서 데이터에 UBCF, IBCF 모델을 적용할 수 있도록 전처리
2. 도서에 대해 train 데이터를 바탕으로 UBCF, IBCF 모델로 test 데이터를 예측
    - 이웃의 개수를 의미하는 nn을 수정하며 MAE가 최소가 되는 최적 모델 설정
    - train 데이터로부터 가장 MAE가 낮은 모델을 test에 적용하여 결과 확인 및 비교
3. 영화와 도서를 추천하는 각 모델의 성능을 비교해보고, 그 이유를 추론하기

(1) 도서 데이터에 UBCF, IBCF 모델을 적용할 수 있도록 전처리

```
## [1] "train data에서의 유일한 book_code 개수: 5865"
```

```
## [1] "test data에서의 유일한 book_code 개수: 5339"
```

```
## [1] "train과 test 데이터의 교집합 개수: 5339"
```

이번에는 위의 4번에서 영화 평점을 예측하고, 추천해줬던 과정을 도서에 적용해보려고 한다. 이 과정들을 수행하기 위하여 train과 test 데이터에서 유일한 book_code의 개수를 확인해보니, train에서는 5,865개가 있었고, test에서는 그보다 약간 적은 5,339개 나온 것을 확인할 수 있었다. 따라서 최종적으로 intersect 함수로 두 데이터에서 공통으로 속하는 데이터에 대하여 추천 모델을 만들고, 평점을 예측해보도록 한다.

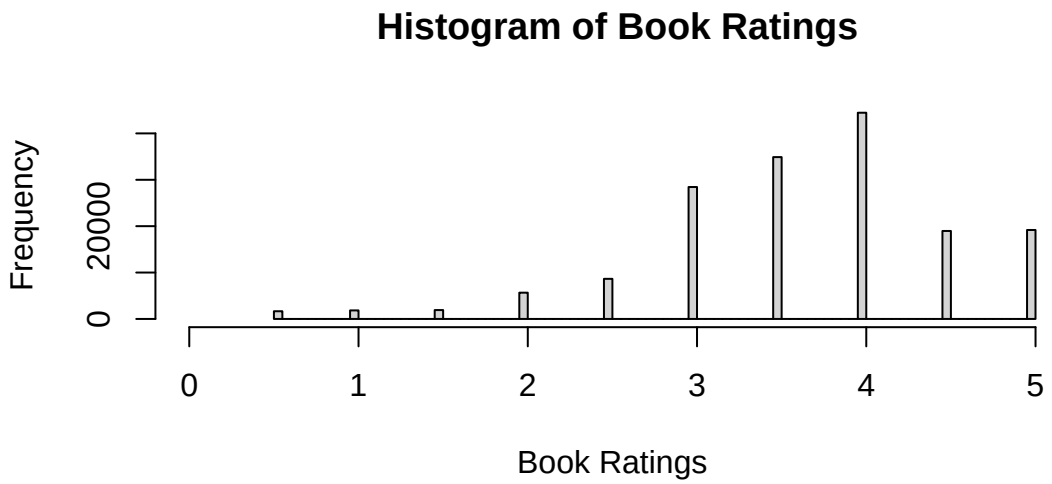
```
## 2361 x 5339 rating matrix of class 'realRatingMatrix' with 165668 ratings.
```

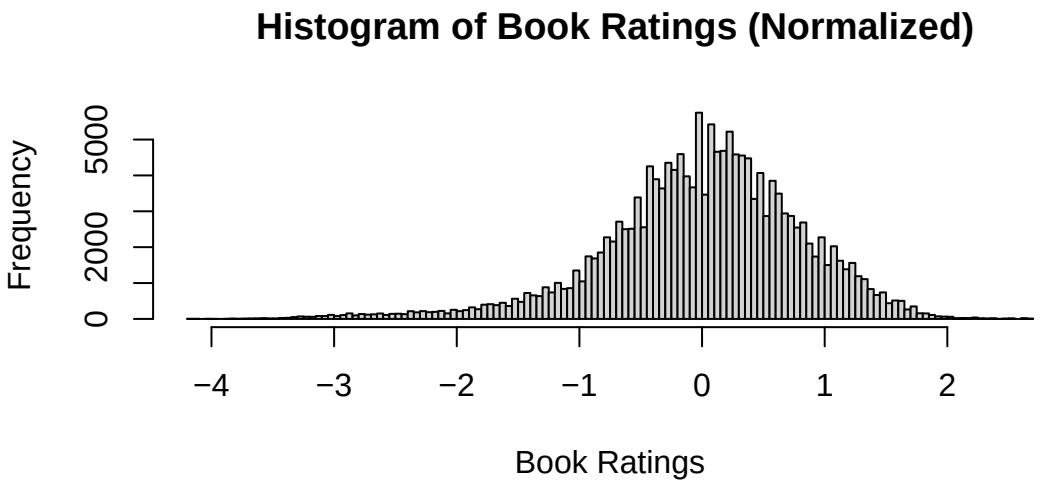
```
## 2361 x 5339 rating matrix of class 'realRatingMatrix' with 42454 ratings.
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    2.00   22.00   47.00   70.17   92.00  670.00
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1.00    7.00   12.00   31.03   27.00  803.00
```

책의 코드와 평점을 바탕으로 spread 하여 row는 사용자가, column에는 도서가 들어가게 하고, 각 셀에는 해당 사용자가 각 도서에 대해 입력한 평점이 들어가도록 한다. 그리고 추천 시스템에 활용될 수 있도록 데이터의 구조를 realRatingMatrix로 변환해주도록 한다. 이후에 결과를 확인하면, train과 test 데이터에서 총 2,361명의 사용자와 5,339개의 책에 대한 평점이 담긴 데이터를 확인할 수 있다. 그리고 train 데이터에 대하여 rowCounts와 colCounts의 summary를 통해 사용자 중에서 가장 많이 책을 본 사람은 총 670개의 책을 봤고, 책들 중에서 총 803명의 사용자가 평점을 입력한 책이 가장 사용자들에게 인기 있었음을 확인할 수 있다. 이후에는 도서에 대하여 사람들이 평점을 어떻게 부여하고 있는지 그 분포를 히스토그램으로 확인해보도록 한다.





사용자들에게 도서를 추천해주기 위하여 현재 train 데이터에서 전반적인 평점의 분포는 어떻게 되는지 히스토그램을 통해 확인해보도록 한다. 가장 먼저 눈에 띄는 것은 앞선 영화와 다르게, 도서들의 평점이 대부분 3부터 5에 많이 속해있음을 확인할 수 있다. 이러한 내용들을 바탕으로 책을 추천해줄 수 있는 모델을 만들고, MAE 등 평가 지표를 비교한 후에 최종적으로는 영화와 도서 중에서 어떤 모델이 더 잘 동작하고 있고, 그렇게 된 이유는 무엇일지 추론해보도록 한다.

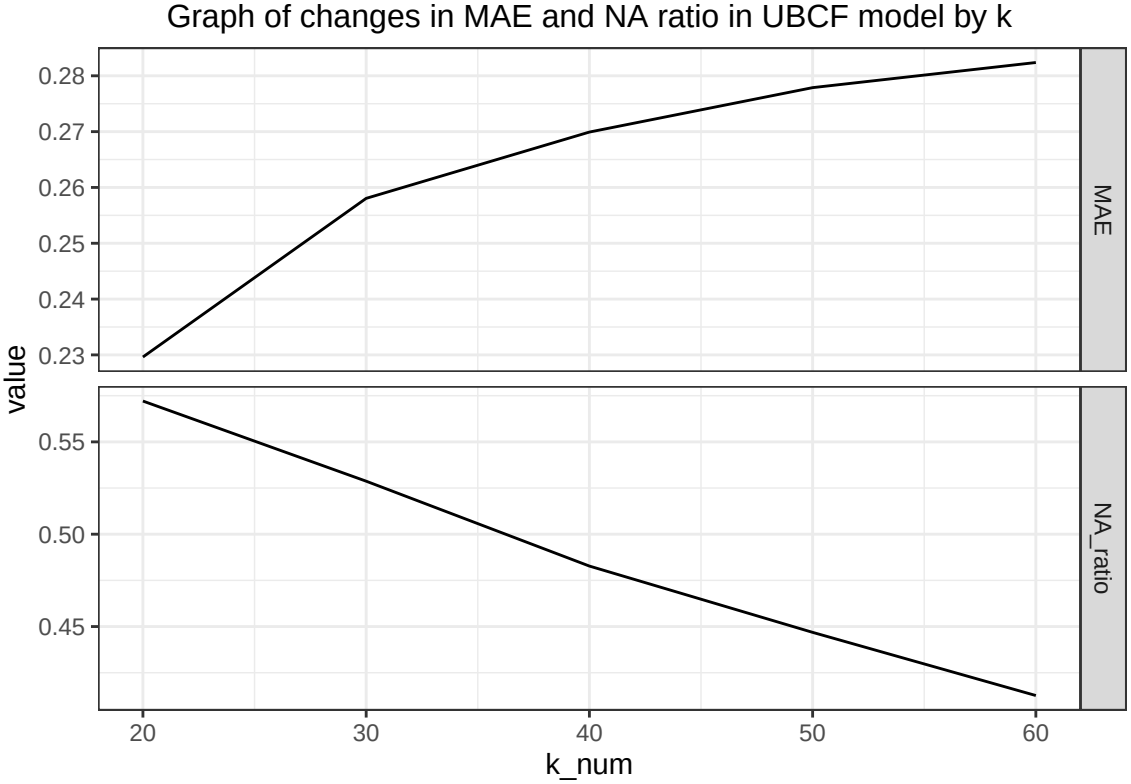
(2) 도서에 대해 train 데이터를 바탕으로 UBCF, IBCF 모델로 test 데이터를 예측

```
##### UBCF model
UBCF_model <- Recommender(book_rrm, method = 'UBCF')

train_pred <- predict(UBCF_model, book_rrm, type = 'ratingMatrix')
train_pred <- as(train_pred, 'data.frame')
colnames(train_pred) <- c('user_code', 'book_code', 'score_pred')

UBCF_book <- book_test %>%
  left_join(train_pred, by = c('user_code', 'book_code')) %>%
  as.data.frame()

UBCF_book %>%
  summarise(MAE = mean(abs(score - score_pred), na.rm = T))
```

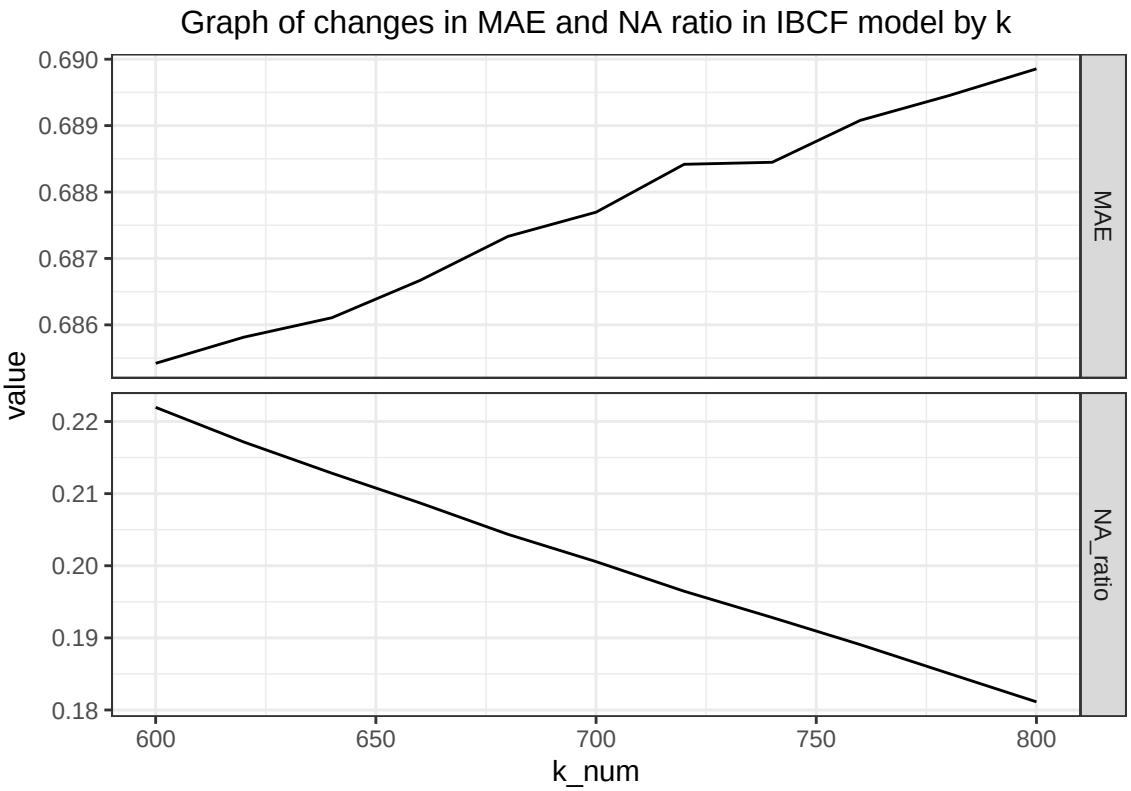
```
##### IBCF model
IBCF_model <- Recommender(book_rrm, method = 'IBCF')

train_pred <- predict(IBCF_model, book_rrm, type = 'ratingMatrix')
train_pred <- as(train_pred, 'data.frame')
colnames(train_pred) <- c('user_code', 'book_code', 'score_pred')

IBCF_book <- book_test %>%
  left_join(train_pred, by = c('user_code', 'book_code')) %>%
  as.data.frame()

IBCF_book %>%
  summarise(MAE = mean(abs(score - score_pred), na.rm = T))
```

```
##            MAE  NA_ratio
## UBCF 0.7492679 0.7377161
## IBCF 0.7476581 0.7357846
```

이번에도 앞선 영화 추천과 동일하게 UBCF, IBCF 모델을 활용하여 각 사용자들의 도서 평점을 예측할 수 있는 추천 모델을 만들어보도록 한다. 전반적인 프로세스는 앞선 문제와 동일하며, 이번 과정에서도 모델의 기본값 parameter로 모델을 만들어서 결과를 비교해보도록 한다. 이 과정들을 수행한 코드는 위와 같음을 확인할 수 있고, 도서에 대한 train, test 데이터에 대하여 realRatingMatrix로 변환하고, Recommender를 통해서 각각 모델을 만들어주도록 한다. 그 후, 최종적으로 train 데이터로부터 찾은 최적의 모델에서 test 데이터의 결과는 어떠한지 확인해본다. 그 결과를 보면, 앞선 영화 추천 모델과 다르게 default parameter로 만든 모델이지만, 상대적으로 NA 비율이 소폭 감소했고, UBCF와 IBCF 모델의 결과가 거의 유사한 것을 확인할 수 있다. 이번 과정에서도 이웃의 개수를 조정하면서 최종적인 모델의 성능을 다시 확인해보고자 한다.



영화 추천 과정과 거의 유사한 코드 작업과 결과이므로, 이번에는 코드를 생략하고 최종적인 결과만 확인해보도록 한다. 먼저 UBCF 모델에서 결과를 확인하면, 20부터 60까지 변화하는 이웃의 개수에 따라 MAE와 NA ratio를 그래프로 표현해보았다. k의 값을 조금이라도 조정해보니, default 값 일때에 비해 성능들이 전반적으로 좋아진 것을 확인할 수 있다. 기존에는 MAE가 0.75이고, NA ratio가 0.74였는데 이웃의 개수를 조정하고 나서는 전반적으로 MAE는 0.2~0.3 정도로, NA ratio는 50% 이하로 떨어진 것을 확인할 수 있다. 현재 나온 결과로만 봐서는 감소하는 NA ratio에 대해 MAE 값이 증가하는데, 그나마 k가 30일때, 그 증가폭이 낮아서 이때가 최적이라고 할 수 있겠다. 이에 대한 test 데이터에 대한 결과는 아래 IBCF 모델과 함께 확인하도록 한다.



이번에는 IBCF 모델에서의 성능을 확인해보도록 한다. UBCF 모델과 다르게 전반적으로 높은 k 값으로 모델을 만들어보았다. 값의 범위 차이가 심하다보니 UBCF, IBCF 모델 각각을 비교할 수 있을지 모르겠지만, 모델을 학습하는데 있어서 오랜 시간이 걸려서 불가피하게 현재 완성된 모델로 그 결과를 확인 및 비교하도록 한다. IBCF 모델 역시 UBCF와 동일하게 k의 값이 증가함에 따라 NA ratio는 줄어들고, MAE는 상승하는 것을 볼 수 있다. 결과를 확인하며, 이 모델에서 최적의 성능을 가지는 k를 보면, 약 740 일때, MAE가 증가하다가 소폭 감소하는 형태를 보인다. 따라서 IBCF 모델에서의 최적 모델은 k가 740 일때로 설정하여 최종적으로 UBCF와 IBCF 모델로 test 데이터에 대해 MAE와 NA ratio를 비교하면 어떻게 되는지 확인해본다.

```
##      k_num       MAE  NA_ratio
## UBCF    30 0.7465328 0.7137608
## IBCF   740 0.7060706 0.1340510
```

앞서 이야기했듯, k 값의 범위 차이가 조금 컸던 이유인지 각 모델들을 비교하는데 어려움이 있었다. 우선 현재 나와있는 결과만 보면, 비교적 낮은 이웃의 개수를 갖는 UBCF 모델은 아무래도 충분히 유사한 사용자를 확인하지 못해서 그런지 NA의 비율이 상당히 높았고, MAE 값은 0.74가 나왔다. 반대로 비교적 많은 이웃의 개수를 가졌던 IBCF 모델은 충분한 데이터를 보았기 때문에 NA의 비율은 확연히 낮았고, 그에 따라 MAE는 약 0.71 정도가 나왔다. 두 모델들이 NA 비율은 상당한 차이를 갖는 반면, 생각보다 MAE 값은 그 편차가 심하지 않았는데, 확실하지는 않겠지만 당장의 MAE는 UBCF 모델이 높긴 하겠으나 k의 개수, NA의 비율 등의 상황을 고려해보면 UBCF 역시 낮은 성능은 아닐 것 같다.

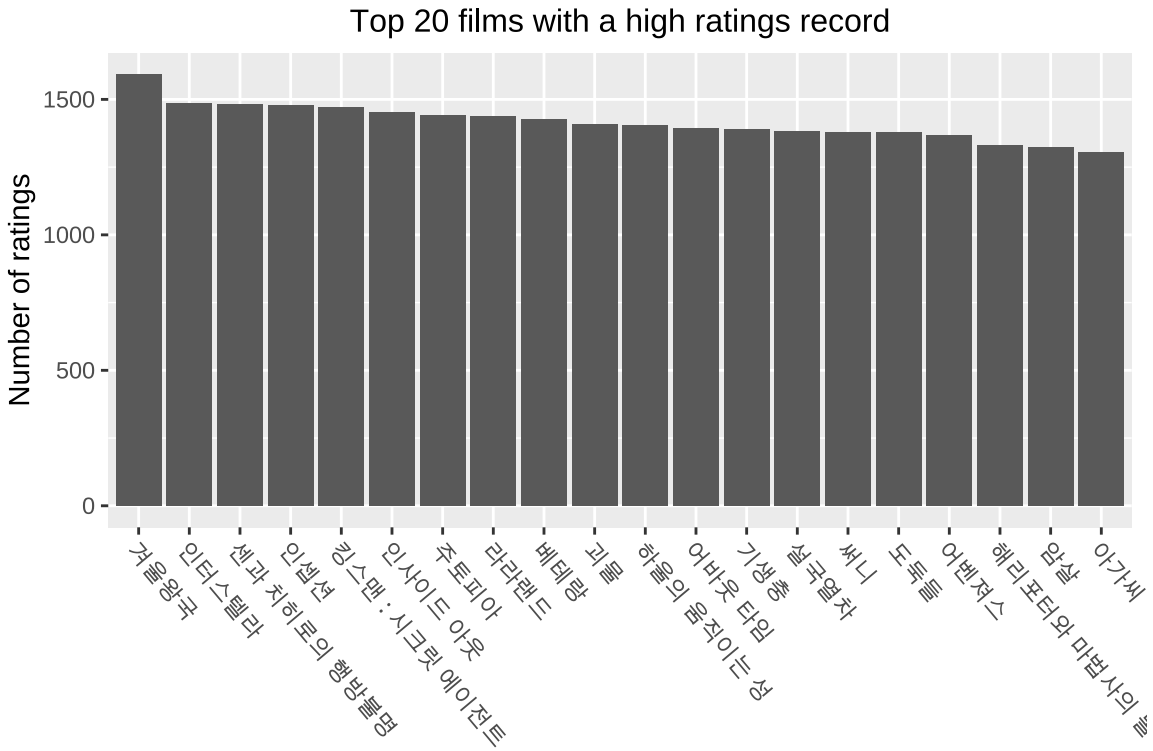
최종적으로 영화와 도서 추천을 하면서 어떤 데이터에 대하여 모델이 더 좋은 성능을 보였는지 비교 및 추론을 해보려고 한다. 안타깝게도, 현재 가지고 있는 성능 지표는 MAE 밖에 없어서 폭넓은 비교가 어려웠다. 시간적 여유가 된다면, 영화와 도서 역시 평균적으로 3점대 후반의 평점을 가지고 있으므로 이 점수를 가지고 binary 문제로 바꿔서 precision이나 recall 등의 성능 지표로 비교해볼 수 있을 것 같다. 일단

현재까지 나온 결과로 모델의 성능을 비교해보면, 영화와 도서 모두 default parameter를 사용했을때는 영화에 비해 도서 데이터에 대해 UBCF와 IBCF 모델 모두 NA ratio가 낮았다. 한편, MAE 지표에 대해서는 그 이유에 대한 깊은 해석이 필요한데, 영화의 경우에는 UBCF 모델은 상당히 좋은 성과를 보였던 것에 반해 IBCF는 전반적으로 성능이 좋지 못했다. 도서는 UBCF와 IBCF 모델의 성능이 거의 비슷했는데, 이 이유에 대하여 고민해보니, 가장 먼저 사용자와 데이터의 수를 생각해보았다. 영화는 2330명의 사용자에 대해 총 10815개의 영화 데이터로 평점을 예측했는데, 아무래도 평점을 확인할 수 있는 영화 데이터가 충분하다보니 특정 사용자로부터 다른 사용자들의 전반적인 특징들을 더 풍부하게 비교할 수 있었을 것이다. 하지만 도서의 경우에는 2381명의 사용자에 대하여 총 5339개의 도서를 보다보니 상대적으로 영화보다는 그 특성들을 UBCF 모델로 찾는게 제한적이었을 것이라고 생각한다. 따라서 추천 모델을 학습하는 데이터 등의 이유로 UBCF 모델에 있어서 도서보다 영화에서 추천 성능이 더 뛰어났다고 이야기할 수 있겠다.

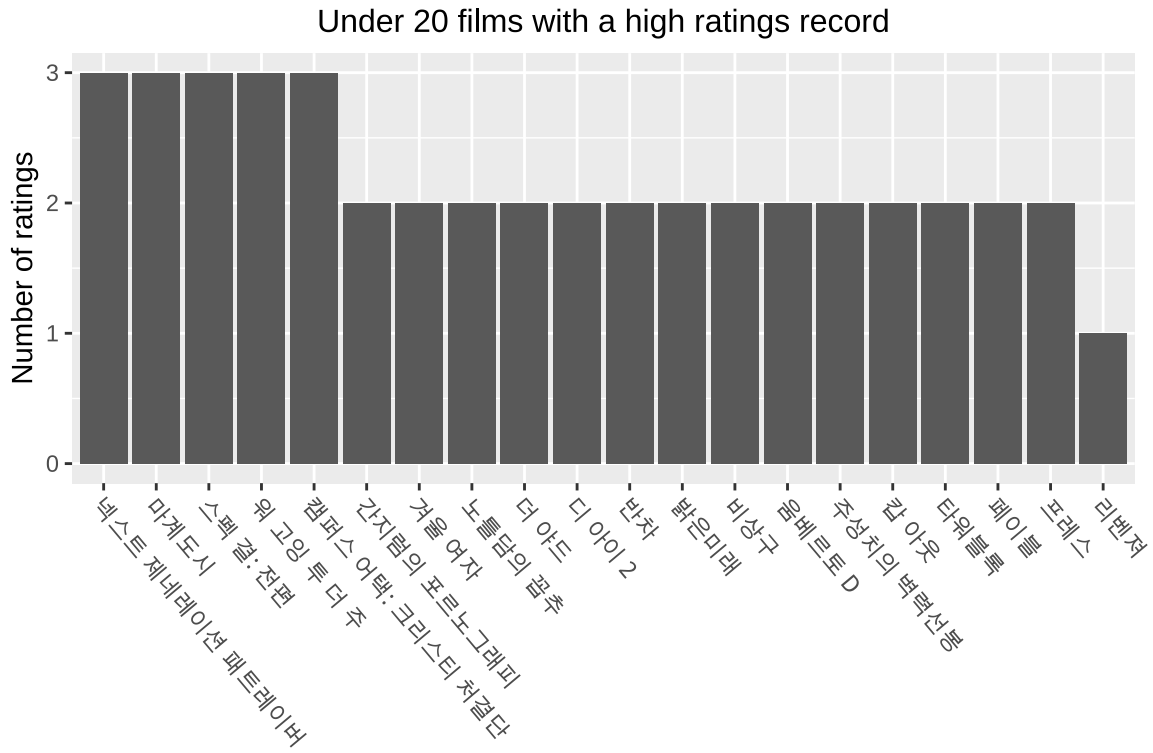
Task 6. Self 추천

1. 영화 데이터에 대하여 self 추천 모델 만들기
2. 도세 데이터에 대하여 self 추천 모델 만들기
    - 사용자들이 평점을 많이 남긴 영화 중에서 20개를 추려서 모델 만들기
    - 상대적으로 사용자들이 평점을 적게 남긴 영화 중 20개를 추려서 모델 만들기

(1) 영화 데이터에 대해 self 추천 모델 만들기



`self` 추천을 할 수 있도록 내가 직접 봤던 영화와 도서 20개에 대해 평점을 임의로 입력하고, 그에 따라 어떤 추천 결과가 나오는지를 확인해보려고 한다. 이를 위해서 두 가지 시나리오로 모델을 만들어보고자 한다. 먼저 첫번째는 전반적으로 사용자들의 많은 관심을 받았던 즉, 평점 기록이 많은 영화를 대상으로 모델을 만들어서 결과를 확인하도록 한다. 두번째로는 상대적으로 관심이 적었던 영화들을 선택하여 평점을 매겨 결과를 확인해보고자 한다. 이를 위해서 먼저 영화 별로 평점 개수에 따른 순서로 시각화를 해보았다. 상위 20개 영화는 모두 봤던 영화였기 때문에 이들을 선택하여 평점을 매긴 후 모델을 만들어볼 것이다.



한편, 하위 20개 영화들 중에서는 처음 보는 것들도 있었기 때문에 평점 기록이 적은 순으로 정렬한 후, 20개를 추려서 결과를 확인해보도록 한다.

```
# Self recommendation - UBCF model
self_movie <- movie_train %>%
  group_by(movie_code) %>%
  mutate(n = n()) %>%
  filter(row_number() == 1) %>%
  arrange(desc(n)) %>%
  head(20) %>%
  select(-user_code, -score, -n) %>%
  mutate(user_code = 'US000000') %>%
  select(user_code, movie_code, movie_title)

self_movie$score <- c(4.5, 4, 2, 2.5, 5, 4.5, 3, 5, 4, 4,
                      2, 4, 4, 2, 3, 4, 5, 5, 5, 3)

self_movie_matrix <- self_movie %>%
  select(-movie_title) %>%
  spread(movie_code, score) %>%
  remove_rownames() %>%
  column_to_rownames(var = 'user_code')

self_movie_rrm <- as(as(self_movie_matrix, 'matrix'), 'realRatingMatrix')

self_UBCF_model <- Recommender(self_movie_rrm, method = 'UBCF')
train_pred <- predict(self_UBCF_model, self_movie_rrm[1], type = 'topNList')
train_pred <- as(train_pred, 'vector')
```

```
##    movie_code                          movie_title
## 1   MC0000163                        내 이름은 칸
## 2   MC0000376                        너의 이름은.
## 3   MC0001836                             시카고
## 4   MC0001856 FYRE: 꿈의 축제에서 악몽의 사기극으로
## 5   MC0001886                미국 수정헌법 제13조
## 6   MC0002164                          섹스토피아
## 7   MC0003912                             첨밀밀
## 8   MC0003921                             고지전
## 9   MC0005843                       살아남은 아이
## 10  MC0009153                          이카로스
```

가장 먼저 전반적으로 많은 사용자들에게 관심도가 높았던 영화를 선정하도록 한다. 위에서 제시한 상위 20개 영화 모두 한번씩은 봤던 영화였으므로, 이들만 추려서 user_code에 US000000을 넣어서 나라는 사용자를 정의해주도록 한다. 그리고 각 영화들에 대한 나의 평점을 넣어서 데이터 프레임을 만든 후에, 기존에 movie_train 데이터와 bind 하여 최종적인 영화 데이터 파일을 만들어준다. 그 후에는 앞선 문제들에서 했던 방식과 동일하게 row 단위를 사용자로, column 단위를 영화로 하는 matrix를 만들고, 추천 모델을 만들 수 있도록 realRatingMatrix 데이터 구조로 변환한다. 그 후, UBCF와 IBCF 모델을 만들어서 각각의 추천 결과를 확인하도록 한다. UBCF 방식을 통해 나온 결과를 확인해보니 알고 있는 영화는 내 이름은 칸, 너의 이름은 이라는 두 영화였는데, 이 영화들은 개인적으로 재밌게 봤던 영화인 것 같다. 나머지 영화들은 잘 알지 못해서 줄거리를 조금 찾아보았는데, 시카고는 뮤지컬 영화로 음악과 함께 영상을 즐길 수 있는 재밌는 영화라고 생각했고 라라랜드라는 영화와 유사한 것으로 생각되었다. 첨밀밀이라는 영화는 조금 오래되긴 했지만 홍콩 멜로 및 로맨스 영화로 어바웃타임과 같은 영화를 좋아했던 사용자로부터 추천을 받지 않았을까 생각해본다.

```
##    movie_code      movie_title
## 1   MC0006324     바람이 불 때
## 2   MC0007973      맛있는 그대
## 3   MC0008783    브롱스 이야기
## 4   MC0011369         바이오맨
## 5   MC0009445 세상 끝과의 조우
## 6   MC0014551 진짜 진짜 잊지마
## 7   MC0010983      북경 자전거
## 8   MC0016390        몽소 빵집
## 9   MC0020173   도라 도라 도라
## 10  MC0014818 포스 오브 네이처
```

앞선 UBCF 모델과 동일한 방법으로 모델만 IBCF로 바꿔서 똑같은 결과를 확인해보았다. 이번 결과는 UBCF 모델보다 더 잘 알려지지 않았던 영화들이 나왔는데, 줄거리만 봐서는 영화의 분위기나 스토리를 알기 조금 제한적이여서 이 추천이 합리적인가 하는 질문에 대한 답은 정확히 내리는 것이 어려웠다. 다만 내가 선택한 영화들 즉, item들과 비슷한 영화로 추천이 되었다고 하니 나중에 기회가 되면 보고 싶다는 생각이 들었다.

```r
# 비교적 평점이 많지 않았던 영화들을 선택하여 추천 모델 만들기
self_movie <- movie_train %>%
  filter(movie_code %in% c('MC0005482', 'MC0004177', 'MC0003414',
                           'MC0011700', 'MC0006955', 'MC0007115',
                           'MC0002720', 'MC0003692', 'MC0007330',
                           'MC0004079', 'MC0005392', 'MC0003385',
                           'MC0000317', 'MC0003264', 'MC0005114',
                           'MC0004195', 'MC0004926', 'MC0003048',
                           'MC0004047', 'MC0004196')) %>%
  group_by(movie_code) %>%
  filter(row_number() == 1) %>%
  mutate(user_code = 'US000000') %>%
  select(-score)

self_movie$score <- c(4, 3, 3, 4, 5, 2, 3, 3, 4, 4, 3, 5, 5, 3, 5, 4, 5, 4, 3, 3)
self_movie
```

```
## # A tibble: 20 x 4
## # Groups:   movie_code [20]
##    user_code movie_code movie_title                score
##    <chr>     <chr>      <chr>                      <dbl>
##  1 US000000  MC0000317  커뮤터                         4
##  2 US000000  MC0002720  성난 황소                      3
##  3 US000000  MC0003264  명탐정 코난 : 14번째 표적      3
##  4 US000000  MC0003385  슈퍼맨                         4
##  5 US000000  MC0003414  007 골든 아이                  5
##  6 US000000  MC0003692  쏘우                           2
##  7 US000000  MC0003048  매드 맥스                      3
##  8 US000000  MC0004047  바람의 검심 : 추억편           3
##  9 US000000  MC0004079  악질경찰                       4
## 10 US000000  MC0004177  에이리언 VS 프레데터 2         4
## 11 US000000  MC0004195  무서운 영화 3                  3
## 12 US000000  MC0004196  삼국지 - 용의 부활             5
## 13 US000000  MC0005114  벤허                           5
## 14 US000000  MC0005392  고스트버스터즈 2               3
## 15 US000000  MC0005482  삼국지: 명장 관우              5
## 16 US000000  MC0004926  골든 슬럼버                    4
## 17 US000000  MC0007115  S.W.A.T - 특수기동대           5
## 18 US000000  MC0007330  미스터 주: 사라진 VIP          4
## 19 US000000  MC0006955  대장 김창수                    3
## 20 US000000  MC0011700  47미터 2                       3
```

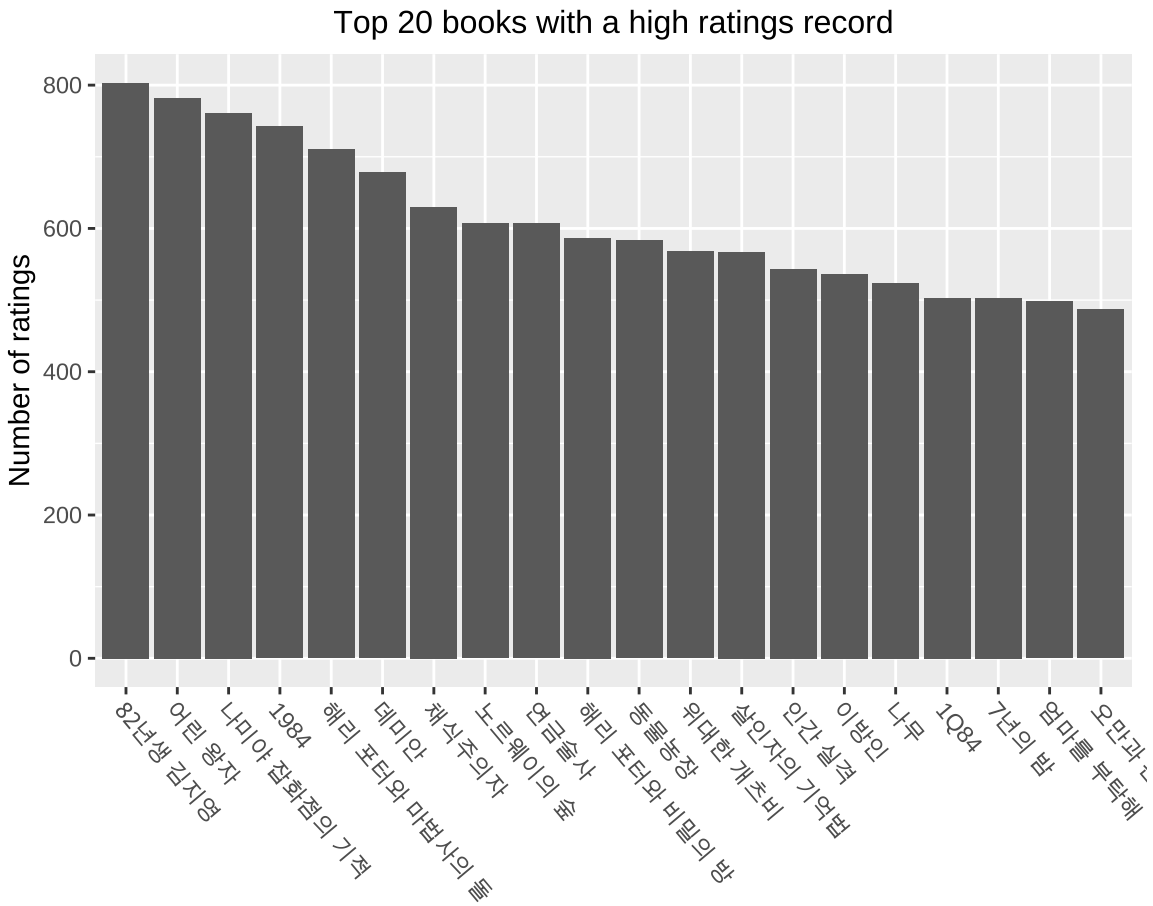
```
##    movie_code                                    movie_title
## 1   MC0001671                                     화니 페이스
## 2   MC0001862 보랏 - 카자흐스탄 킹카의 미국 문화 빨아들이기
## 3   MC0002511                                   핑크 플라밍고
## 4   MC0005815                                      신은 없다
## 5   MC0005962                극장판 암살교실: 365일의 시간
## 6   MC0006082                                       오디션
## 7   MC0007088                                   러브 익스포져
## 8   MC0004745                                    이치 더 킬러
## 9   MC0003282                                    영원한 여름
## 10  MC0004936                                       레이드 2
```

이번에는 사용자들의 평점이 100개가 넘지 않았던 비교적 인기가 없는 영화들을 선택하여 UBCF 추천 모델을 만들어보도록 한다. 영화 코드로 filtering을 한 후에 사용자 코드를 US000000으로 설정하여 데이터를 만들어준다. 그리고 앞선 과정과 동일하게 모델을 만들어서 결과를 확인해보면 다음과 같은 결과를 얻을 수 있다. 앞서 전반적으로 평점이 많았던 데이터에서 추천 결과에서도 비교적 잘 알려진 영화들이 나오지 않았는데, 이번 결과는 더더욱 숨겨진 영화들이 나왔던 것으로 생각되었다.

```
##    movie_code      movie_title
## 1   MC0001469           세레나
## 2   MC0001490        치킨 리틀
## 3   MC0001557    밀리언 웨이즈
## 4   MC0001676 지난여름, 갑자기
## 5   MC0001730   내가 사는 피부
## 6   MC0001963           허슬러
## 7   MC0001981     비치온더비치
## 8   MC0001982             처음
## 9   MC0002196  어두워질 때까지
## 10  MC0002260         유리정원
```

IBCF 추천 모델 역시 UBCF와 비슷하게 전반적으로 잘 알려진 영화가 나오지는 않았고, 주어진 제한된 데이터 속에서 취향을 내 취향을 반영하는데에는 약간의 한계가 있는 것으로 생각된다.

(2) 도서 데이터에 대해 self 추천 모델 만들기



영화와 동일한 과정을 이번에는 도서에도 적용해보도록 한다. 사용자들의 평점이 많이 담겼던 비교적 인기 있는 도서들을 먼저 확인해보는 과정을 수행했다. 전반적으로 흔히 베스트셀러라고 알려진 영화들이 많이 추출되었고, 1984나 해리포터와 같이 도서 뿐 아니라 영화로도 나온 것들도 많은 사용자들의 관심을 받았다. 그렇다면 도서에 대해서는 어떤 추천 결과를 보이는지 확인해보도록 한다. 현재 출력된 이 20권의 책에 대하여 나의 평점을 매겨서 추천 모델을 UBCF, IBCF로 만들어보도록 한다.

```
# Self recommendation - UBCF model
self_book <- book_train %>%
  group_by(book_code) %>%
  mutate(n = n()) %>%
  filter(row_number() == 1) %>%
  arrange(desc(n)) %>%
  head(20) %>%
  select(-user_code, -score, -n) %>%
  mutate(user_code = 'US000000') %>%
  select(user_code, book_code, book_title)

self_book$score <- c(5, 4, 3, 4, 5, 4, 2, 2, 2, 5,
                     3, 4, 3, 3, 2, 3, 4, 4, 5, 3)

self_book <- bind_rows(self_book[, c('user_code', 'book_code',
                                     'score', 'book_title')], book_train)

self_book_matrix <- self_book %>%
  select(-book_title) %>%
  spread(book_code, score) %>%
  remove_rownames() %>%
  column_to_rownames(var = 'user_code')

self_book_rrm <- as(as(self_book_matrix, 'matrix'), 'realRatingMatrix')

self_UBCF_model <- Recommender(self_book_rrm, method = 'UBCF')
train_pred <- predict(self_UBCF_model, self_book_rrm[1], type = 'topNList')
train_pred <- as(train_pred, 'list')
```

```
##    book_code                  book_title
## 1  BC0000067             호밀밭의 파수꾼
## 2  BC0001633       어서오세요, 305호에!
## 3  BC0001640                 연민의 굴레
## 4  BC0002578              잊기 좋은 이름
## 5  BC0002759      내 어린고양이와 늙은개
## 6  BC0003197  칠면조와 달리는 육체노동자
## 7  BC0004208   아무것도 아닌 지금은 없다
## 8  BC0004361           금요일엔 돌아오렴
## 9  BC0006167 나혜석, 글 쓰는 여자의 탄생
## 10 BC0006660     오늘은 이 바람만 느껴줘
```

```
##    book_code                      book_title
## 1  BC0000071                    영원한 제국
## 2  BC0002188                    합리적 의심
## 3  BC0003296                     내부자들 1
## 4  BC0003301                    도시의 시간
## 5  BC0003531 나의 문화유산답사기 9 : 서울편
## 6  BC0003635                 아름다움의 구원
## 7  BC0004211          하나도 괜찮지 않습니다
## 8  BC0004337                          엠마
## 9  BC0002237                   내려놓음
## 10 BC0003082                 내가 싸우듯이
```

도서 데이터에 대하여 많은 사용자들로부터 관심도가 높았던 도서에 대해 평점을 매겼고, 그에 따른 UBCF와 IBCF 각각에 대한 추천 시스템의 결과이다. 실제로 추천된 도서들 중에서 읽어보거나 들어본 책은 호밀밭의 파수꾼이나 연민의 굴레, 내부자들, 나의 문화유산답사기, 내려놓음 등이었고, 내용들을 잘 알지 못해서 취향이 반영된 추천인지는 확인이 필요할 것 같다고 생각했다.

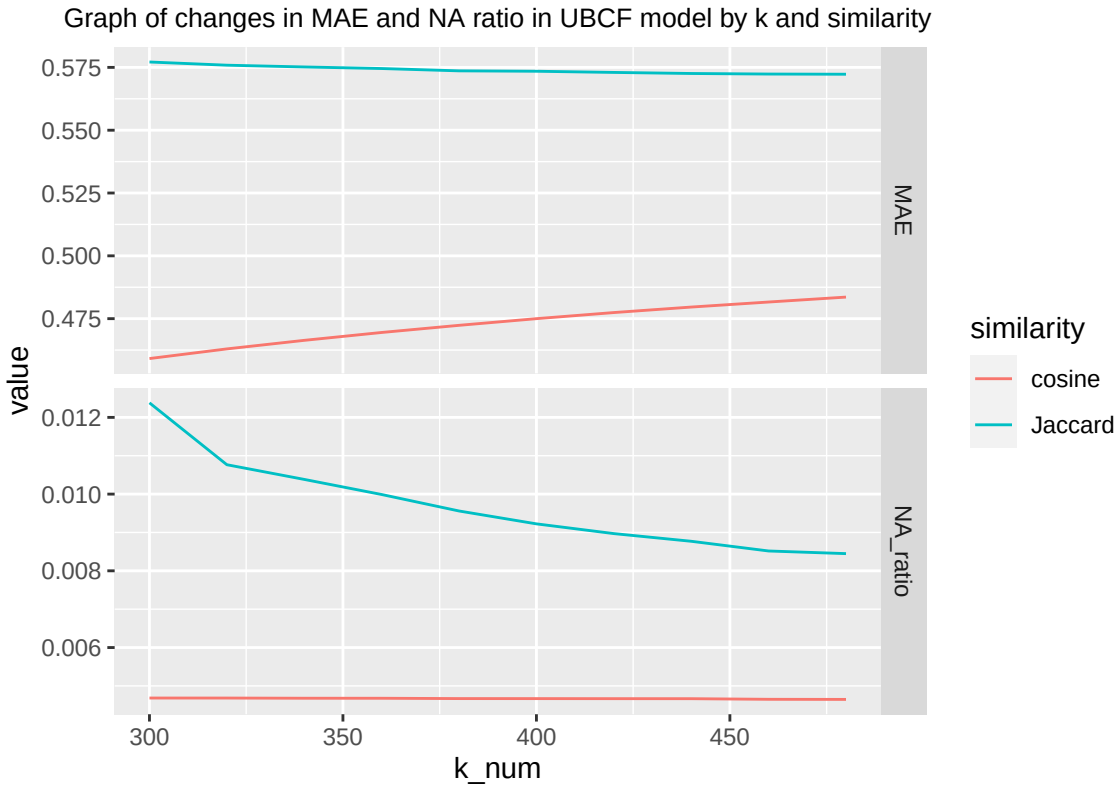
전반적으로 영화와 도서에 대한 self 추천 결과는 주어진 데이터가 한정적이고, 내가 선택한 컨텐츠 역시 많은 수가 아니다보니 각 사용자, 각 컨텐츠 별로의 관계를 비교하는데에는 약간의 어려움이 있었다고 이야기할 수 있겠다. (물론 추천된 컨텐츠를 줄거리 등을 확인해보아도 정확히 잘 알지 못해서 내가 직접 판단하는데에도 어려움이 있었다!)

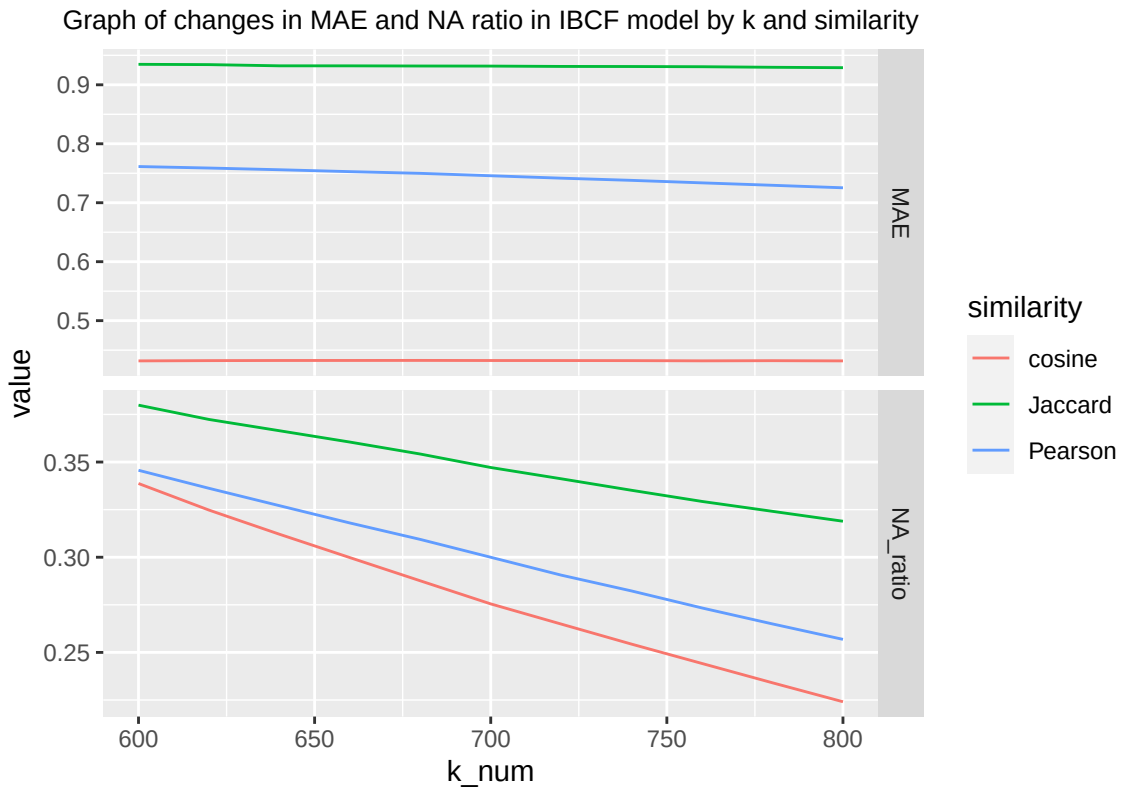
Bonus Task 1. Cross-platform 추천

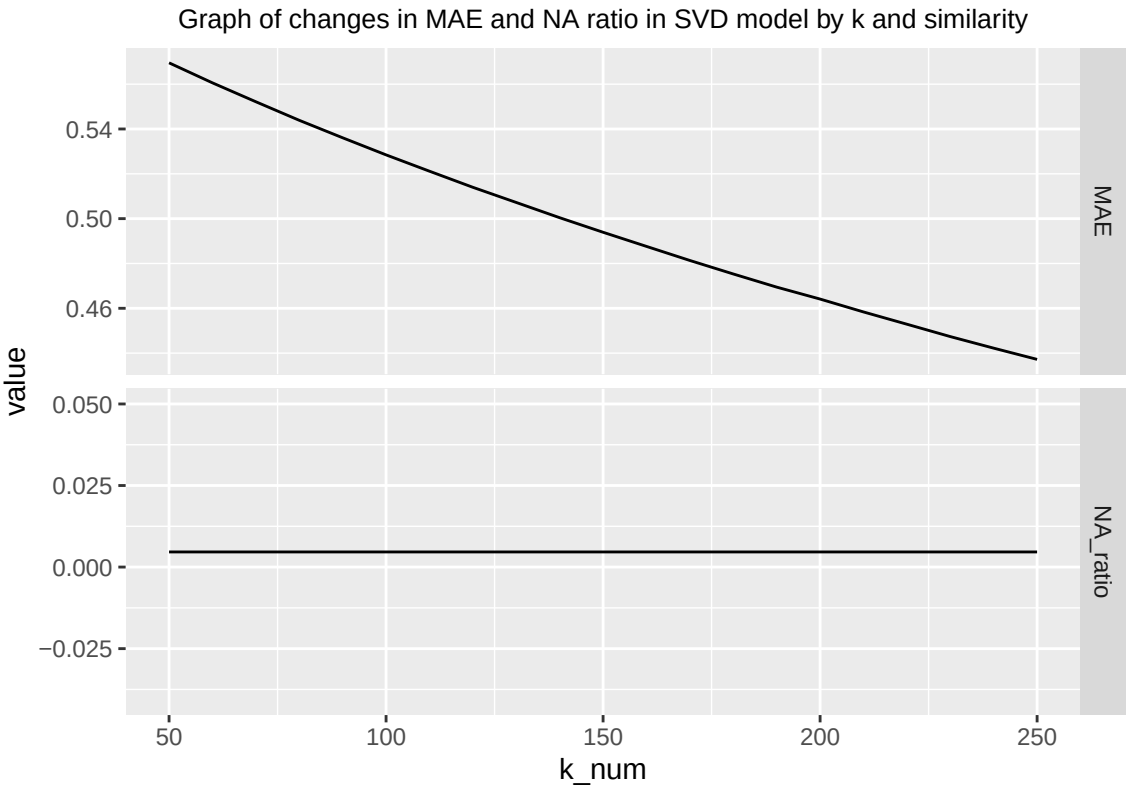
Bonus Task 2. 영화 추천 성능 개선

이번에는 영화 추천 성능을 개선하기 위한 방법론을 고민하고, 실제로 모델을 만들어서 결과를 비교해보고자 한다. 앞선 4번과 5번에서 UBCF, IBCF 모델로 결과를 비교해보았는데, 이는 k 값을 제외하고 recommender의 default parameter들을 활용하여 모델을 만들어보았다. 따라서 이번에는 조금 더 풍성한 모델을 위해서 parameter를 수정하면서 그 결과를 비교해보고자 한다. recommenderRegistry를 통해서 어떠한 모델들이 있는지 그리고 안에 들어갈 수 있는 parameter는 무엇인지 확인해볼 수 있다. 이번 과정에서는 앞서 활용했던 UBCF와 IBCF에서 similarity를 확인할 수 있는 지표가 기존에는 cosine이었는데, 이를 집합 사이에 유사도를 확인할 수 있는 Jaccard 지표로도 결과를 확인해본다. 뿐만 아니라, model-based 추천 방식 중 하나로 알려진 SVD 방식을 사용하여 추천을 수행해보도록 한다. SVD는 특이값 분해 방식으로 Latent Semantic Indexing의 한 분야인데, 해석의 어려움이 있긴 하지만 큰 행렬을 축소시키면서 계산량을 줄이고, 사용자나 아이템 사이의 유사도를 구해준다. SVD 방식은 한 행렬을 세 개의 행렬의 행렬곱으로 표현하는데, 이 문제의 경우에는 사용자 행렬, 영화 행렬 그리고 특이값 행렬로 만들어서 축소된 데이터로 사용자들이 평가하지 않은 영화에 대해서도 평점을 예측한다. 그렇다면 각 모델들의 결과는 어떠할지 확인해보도록 한다.

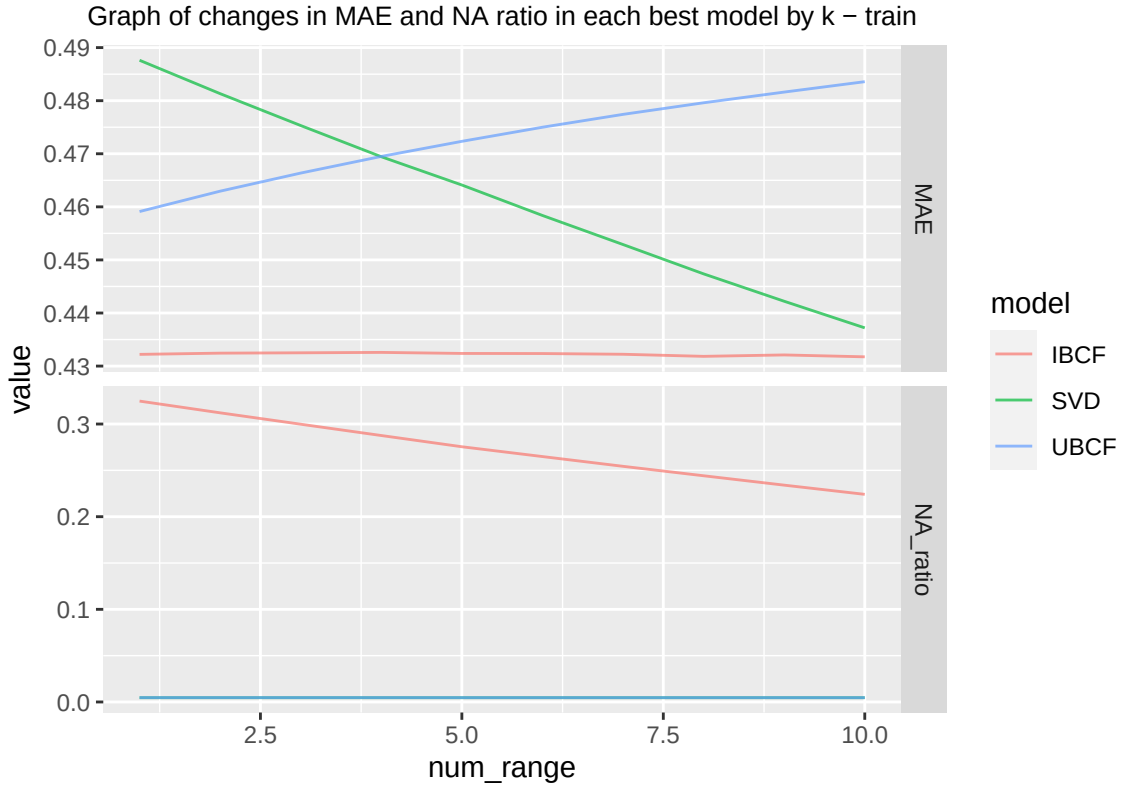
```r
result_df <- data_frame()
for (k in seq(50, 250, by = 10)){
  UBCF_model <- Recommender(movie_rrm, method = 'SVD')
  # parameter(list = c(k = k, method = 'Jaccard', normalize = 'Z-score'))

  train_pred <- predict(UBCF_model, movie_rrm, type = 'ratingMatrix')
  train_pred <- as(train_pred, 'data.frame')
  colnames(train_pred) <- c('user_code', 'movie_code', 'score_pred')

  UBCF_result_test <- movie_test %>%
    left_join(train_pred, by = c('user_code', 'movie_code')) %>%
    as.data.frame()
  UBCF_result_train <- movie_train %>%
    left_join(train_pred, by = c('user_code', 'movie_code')) %>%
    as.data.frame()

  train <- UBCF_result_train %>%
    summarise(MAE = mean(abs(score - score_pred), na.rm = T),
              NA_ratio = mean(is.na(score_pred))) %>%
    mutate(k_num = k, df = 'train', model = 'IBCF', sim = 'cosine', norm = 'center')
  test <- UBCF_result_test %>%
    summarise(MAE = mean(abs(score - score_pred), na.rm = T),
              NA_ratio = mean(is.na(score_pred))) %>%
    mutate(k_num = k, df = 'test', model = 'IBCF', sim = 'cosine', norm = 'center')

  train_test <- bind_rows(train, test)
  result_df <- bind_rows(result_df, train_test)
}
```

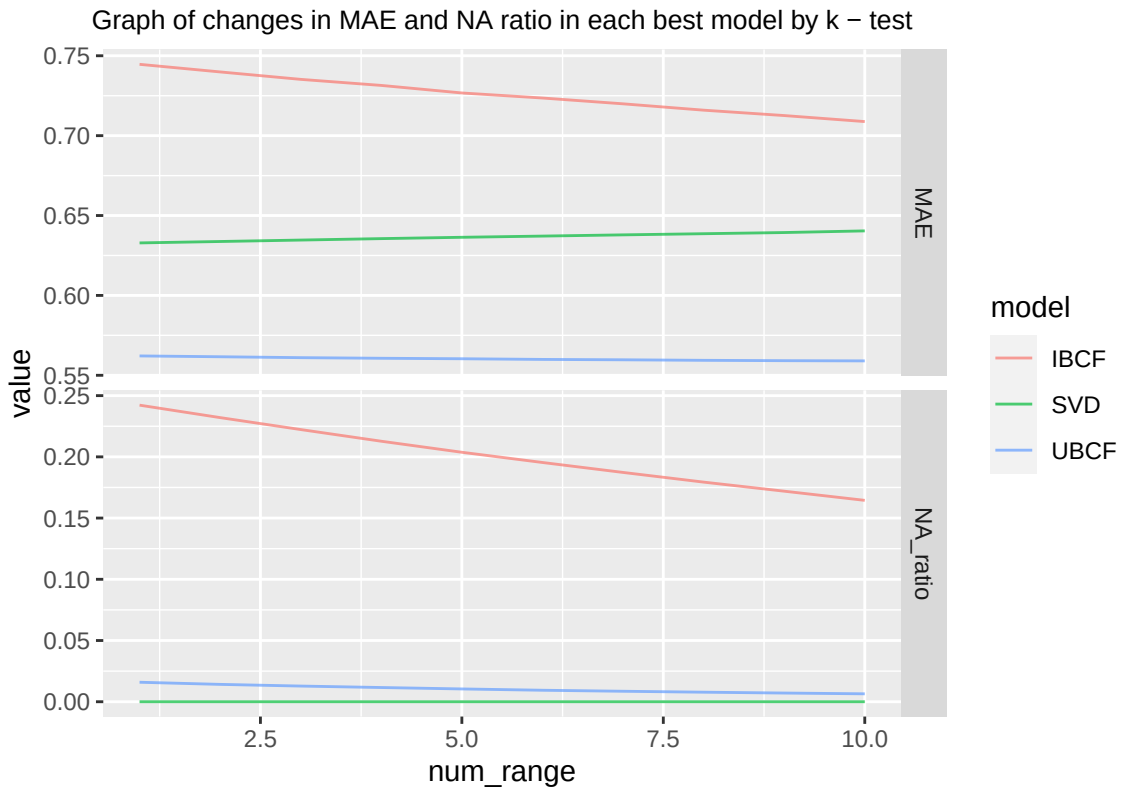



Graph of changes in MAE and NA ratio in IBCF model by k and similarity
value
0.9
0.8
0.7
0.6
0.5
MAE
0.35
0.30
0.25
NA_ratio
600
650
700
750
800
k_num
similarity
cosine
Jaccard
Pearson


Graph of changes in MAE and NA ratio in SVD model by k and similarity
value
0.54
0.50
0.46
MAE
0.050
0.025
0.000
−0.025
NA_ratio
50
100
150
200
250
k_num

모델을 만드는 부분에서는 4번과 5번 과정에서와 유사한 코드를 활용하여 다양한 모델을 만들도록 한다. 앞서 이야기했듯, 이번에는 모델을 만들기 위해서 세부적인 parameter를 수정하려고 하는데, 코드의 주석에서 확인할 수 있듯 parameter에 리스트 형태로 값을 주면 다양한 방법으로 모델을 학습할 수 있다. 사전에 먼저 확인해보니, normalize 부분에서는 default인 center와 Z-score 사이에 큰 차이가 없어서 이번에는 이는 따로 수정하지 않도록 한다. 또한 중간마다 유사도를 측정하는 지표로 Pearson 상관계수가 오류가 나는 부분들도 있어서 일부 모델에서는 이 경우를 제외하고 모델을 만들었다. 각 모델을 학습하는데 꽤 많은 시간이 걸려서 사전에 만들어놓은 결과들을 불러와서 최종적인 결과를 확인해보도록 한다. 성능을 평가하는 지표는 앞서 사용한 MAE와 함께 NA ratio를 확인하도록 한다. 유사도를 평가하는 방법으로 기존 cosine 방법 외에 Jaccard나 Pearson을 활용한 UBCF와 IBCF 모델 그리고 SVD 모델에 대한 결과를 위와 같이 확인할 수 있다. 전반적으로 k의 값이 증가함에 따라 NA의 비율은 줄어들고, MAE는 크게 변화는 없는 것 같다. 다만 SVD 모델은 NA의 비율도 적은데 k가 커지면서 MAE도 감소하는 이상적인 모습을 보이고 있다. 이를 조금 더 직관적으로 확인할 수 있도록 각 모델에서 성능이 가장 좋은 모델들만 추려서 하나의 그래프로 결과를 확인해보도록 한다. 이 경우에는 k의 범위가 모두 다르므로, 비교를 위해서 값의 범위를 통일해주도록 했다.





최종 결과를 확인해본다. 앞서 개별 결과에서도 확인할 수 있었듯, k의 값이 커지면서 NA 비율은 줄어드는데, SVD와 UBCF 모델은 어느 수준 정도 도달하니 거의 NA가 없는 것을 볼 수 있다. 한편, IBCF 모델은 k의 값이 증가해도 MAE 값의 변화가 특별히 없고, UBCF 모델은 오히려 MAE 값이 증가하고 있음을 확인할 수 있다. 그렇다면 새롭게 추가한 SVD 모델은 어떨까? 그래프에서 결과를 확인할 수 있듯, NA의 비율은 적고 계속해서 MAE가 감소하는 것으로 보아 성능이 가장 좋은 모델이라고 이야기할 수 있겠다. 뿐만 아니라, 이 결과를 만들기 위해서 모델을 학습하는 시간을 비교해보더라도, 앞서 이야기했듯 SVD의 결과는 UBCF나 IBCF에 비해 엄청난 속도를 보였다. 계산량도 경제적이면서 그 퍼포먼스도 잘나오는 SVD 모델의 장점을 확인할 수 있었던 것 같다. 하지만 실제 test 데이터에 대한 결과에서는 조금 다른 결과가 나왔는데, 전반적으로 UBCF와 SVD 모델의 MAE 움직임은 거의 없었고, 오히려 SVD 보다 UBCF 모델의 결과가 좋은 것을 볼 수 있다. 모델의 과적합 등을 고려해봐야 할 것으로 보인다.

생각보다 추천 시스템을 학습하는데 걸리는 시간이 오래 걸려서 다양한 시도를 하기에 시간적인 제약이 있었던 것 같다. 교수님께서 말씀해주셨듯, 영화나 도서의 평균 평점을 반영해서 binary 형태로 전반적으로 사용자들이 해당 컨텐츠를 좋아할지, 안할지를 분류하는 문제로 접근하는 방법들도 있을 것 같다. 또한 각 영화 별로 장르, 등급, 관람객 수, 줄거리, 특징, 사용자 후기 등 추가적인 데이터를 수집하여 content-based 모델로도 접근해볼 수 있을 것 같다. 예를 들어, 영화의 줄거리와 특징에 대한 데이터가 있다고 가정했을 때, 특정 사용자가 선호하는 영화군 (평점이 높았던 영화 등)을 알면, 해당 영화들의 줄거리와 특징으로부터 코사인 유사도를 계산해서 모든 영화들에 대하여 유사한 순서를 매겨주고, 가장 순위가 높았던 컨텐츠를 추천해줄 수 있다. 이때, 단순히 줄거리라는 텍스트로 코사인 유사도를 계산하면, 감옥에서 희망을 이야기하는 '7번 방의 선물'과 같은 영화가 감옥이라는 단어 때문에 어두운 영화들과 유사하게 계산될 수도 있다. 따라서 이를 보완하기 위해 영화의 특징들과 같은 추가적인 변수를 활용하여 영화의 분위기도 전반적으로 일치한다면 추천해주는 등의 방법도 고려해볼 수 있을 것이다.