Choose Your Own Project Submission

D-Lorenz

2/1/2021

Overview

This submission for the Capstone Project of the HarvardX Data Science course focuses on the 'Food.com recipes and interactions' dataset, which is publicly available on Kaggle at this address: https://www.kaggle.com/shuyangli94/food-com-recipes-and-user-interactions?select=RAW_recipes.csv. The report follows the directions and code learned troughout the different sections of past courses and the online book by Prof. Irizarry and dslab, as well as the already available notebooks on the kaggle dataset page. This document is conceptually subdivided in two sections: the first section collects the code to perform the preliminary data loading, cleaning, and formatting; the second section dives into the modelling of a predictive system to try to understand what are the main drivers of rating expression based on the available variables of the studied dataset. Due to the length of the report and to avoid breaking up the reading, the results are discussed immediately following the description of the methods used for estimation.

Analysis

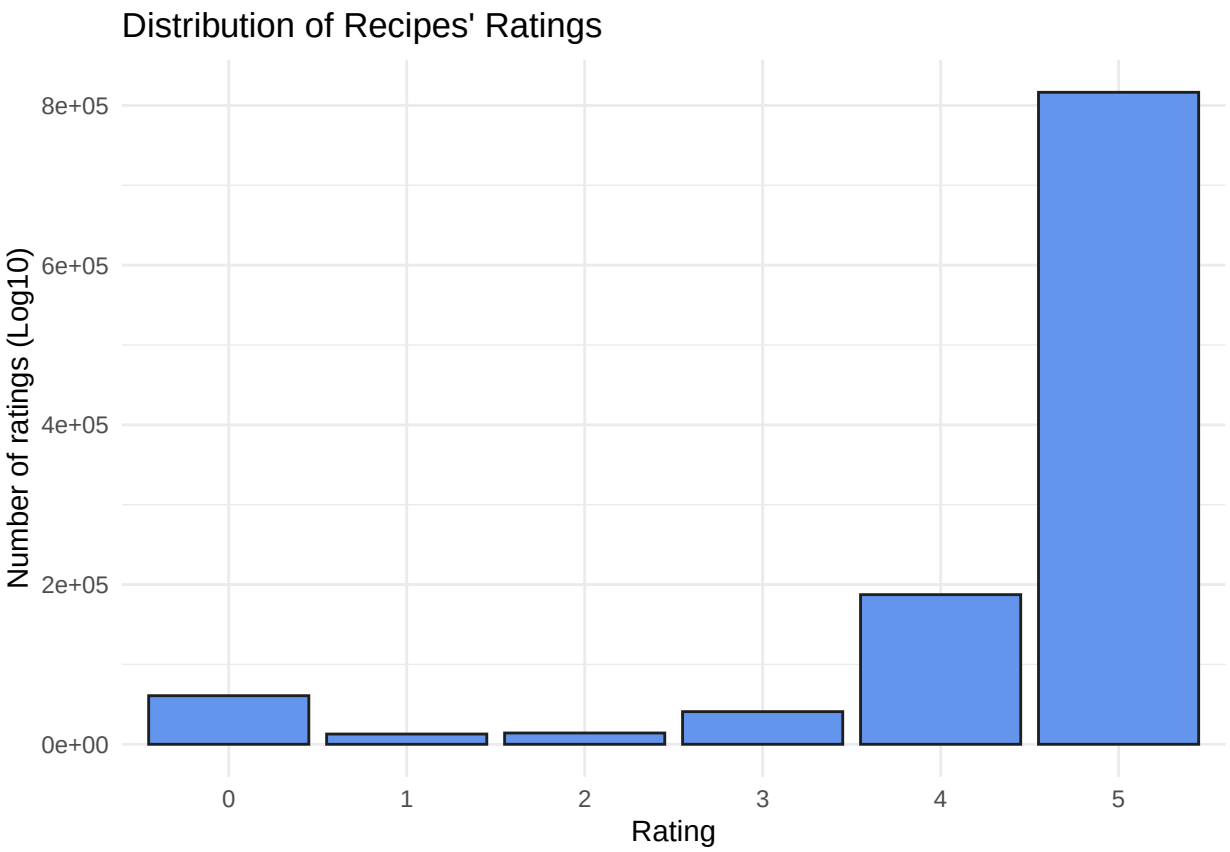
Data preparation

The employed data consist of two separate files, both in csv format: 1) 'RAW_recipes.csv' with 231637 records, it contains the unique recipes entries identified by the tuple: "name" string with recipe name, "id" unique numeric recipe identifier, "minutes" numeric time of preparation, "contributor_id" unique numeric recipe' creator identifier, "submitted" date of submission, "tags" string array with keywords, "nutrition" string array with nutritional attributes (quantity of protein, fat, etc.), "n_steps" numeric number of steps to prepare the food, "steps" string array with the description of each step, "description" string with a general description of the preparation, "ingredients" string array with the list of ingredient, "n_ingredients" numeric count of the employed ingredients; 2) 'RAW_interactions.csv' with 1132367 records, it contains the reviews expressed by the user on the recipes, each record is defined by the tuple: "user_id" unique numeric review' creator identifier, "recipe_id" unique numeric recipe' creator identifier, "date" date of submission, "rating" numeric rating going from 1 to 5, "review" string text of the review. We can already merge these two tables with an inner join on the recipe_id field:

Lets start by looking at the distributions of the recorded variables. The rating field records integer values between zero and five. The zeros stands for not tested recipes, so the records with it would be discarded from the subsequent analyses.

```
data.frame(table(rating = revRec$rating)) %>%
  ggplot()+
  geom_bar(aes(rating, Freq), stat = "identity",
           fill = "cornflowerblue", color = "grey12") +
  theme_minimal()+
  ylab("Number of ratings (Log10)")+
  xlab("Rating")+
```

```
  ggtitle("Distribution of Recipes' Ratings")+
  theme_minimal()
```



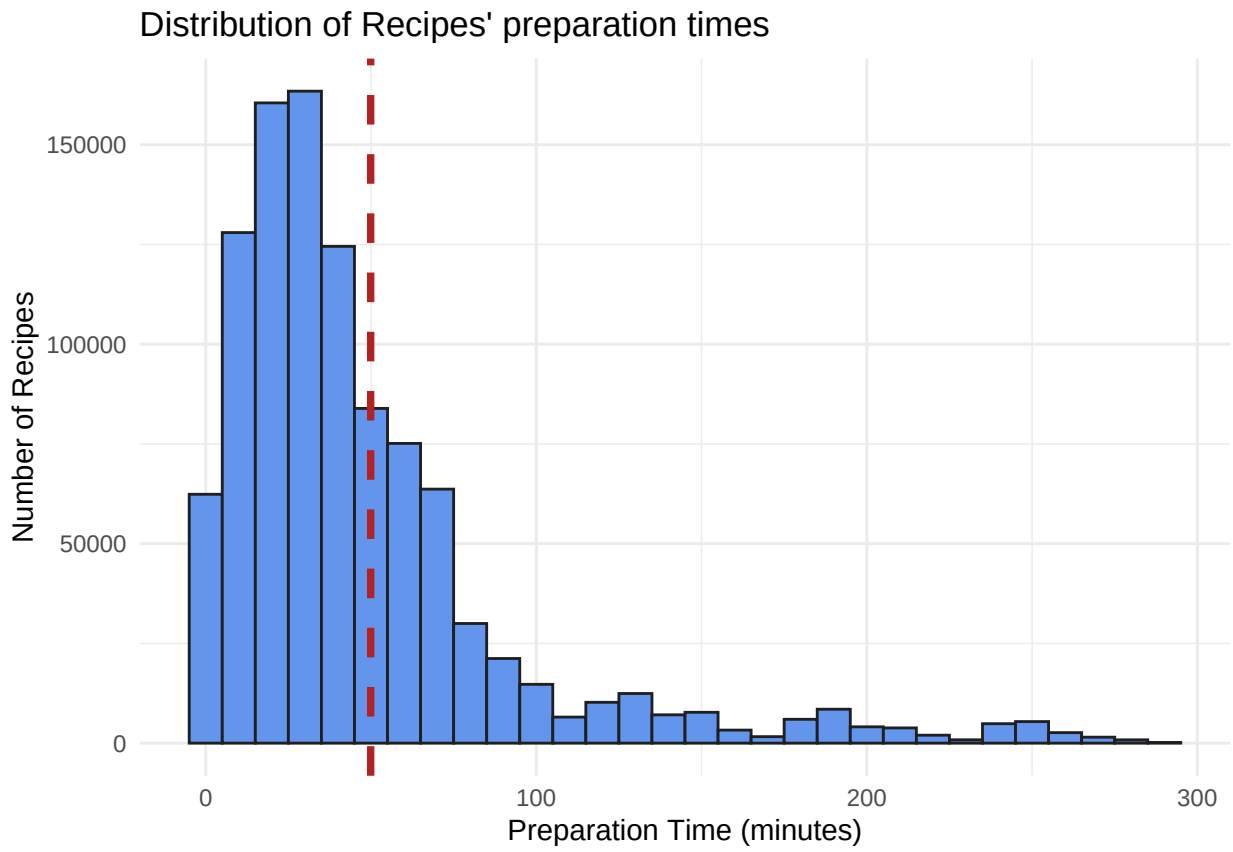

```
# filter out reviews with a rating of 0 that mean no testing of the recipes
revRec <- revRec %>%
  filter(rating > 0)
```

The next numerical fields are 'minutes', 'n_steps', and 'n_ingredients'. The variable with the recipe's preparation length contains the required time to prepare the food according to the specific recipe, and accordingly there should be reasonable times. The average recorded values are around 3.618359×10^4, however there are abnormally large values that skews the distribution, the minutes will be cut at a maximum of 5 hours of preparation time.

```
# filter out reviews with more than 5 hours of preparation time
revRec <- revRec %>%
  filter(minutes < 300)

revRec %>%
  ggplot()+
  geom_histogram(aes(minutes),
                 fill = "cornflowerblue", binwidth = 10, color = "grey12") +
  geom_vline(xintercept = mean(revRec$minutes),
             color="firebrick", linetype = 2, size = 1.3)+
  theme_minimal()+
  ylab("Number of Recipes")+
```

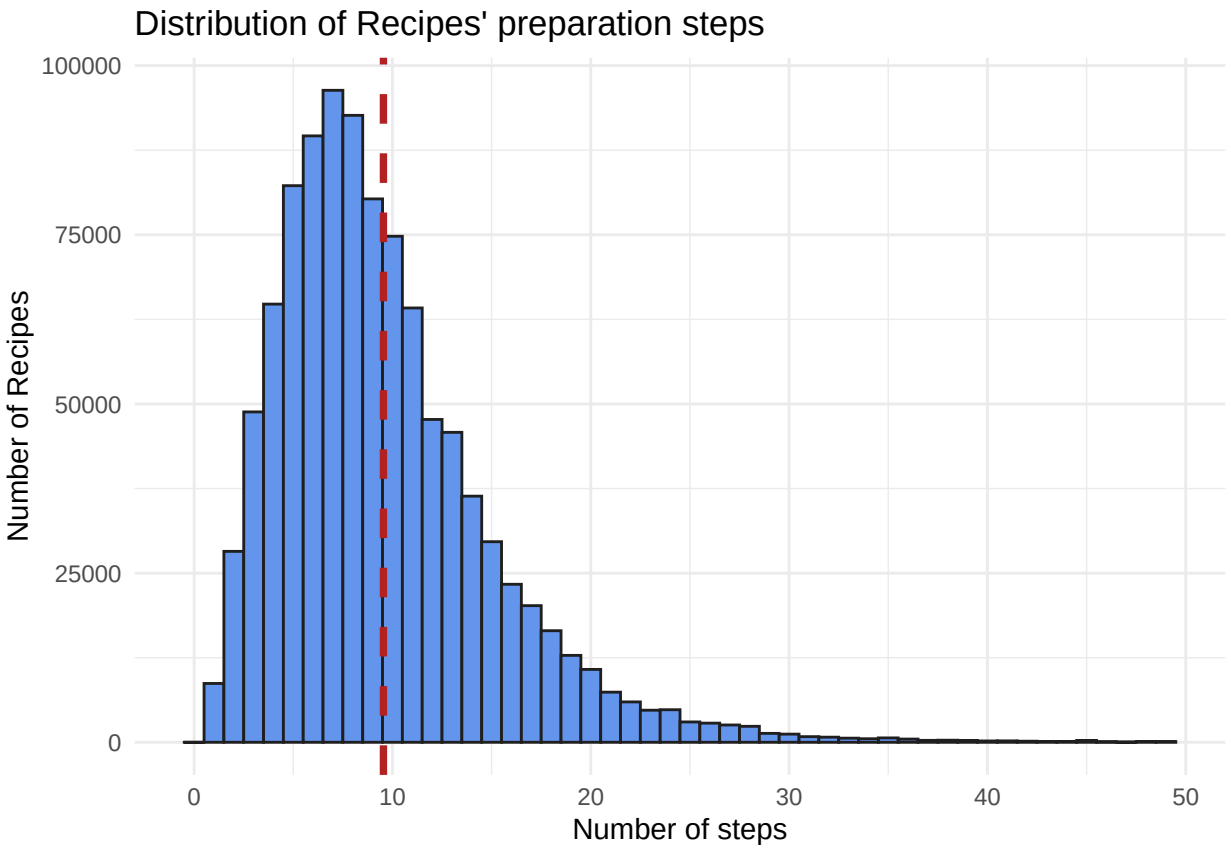
```
  xlab("Preparation Time (minutes)")+
  ggtitle("Distribution of Recipes' preparation times")+
  theme_minimal()
```



The number of steps required to prepare the food according to the specific recipe are stored in the 'n_step' field. The recorded values ranges from 0 to a maximum of 145 steps, the distribution is cut at 50.

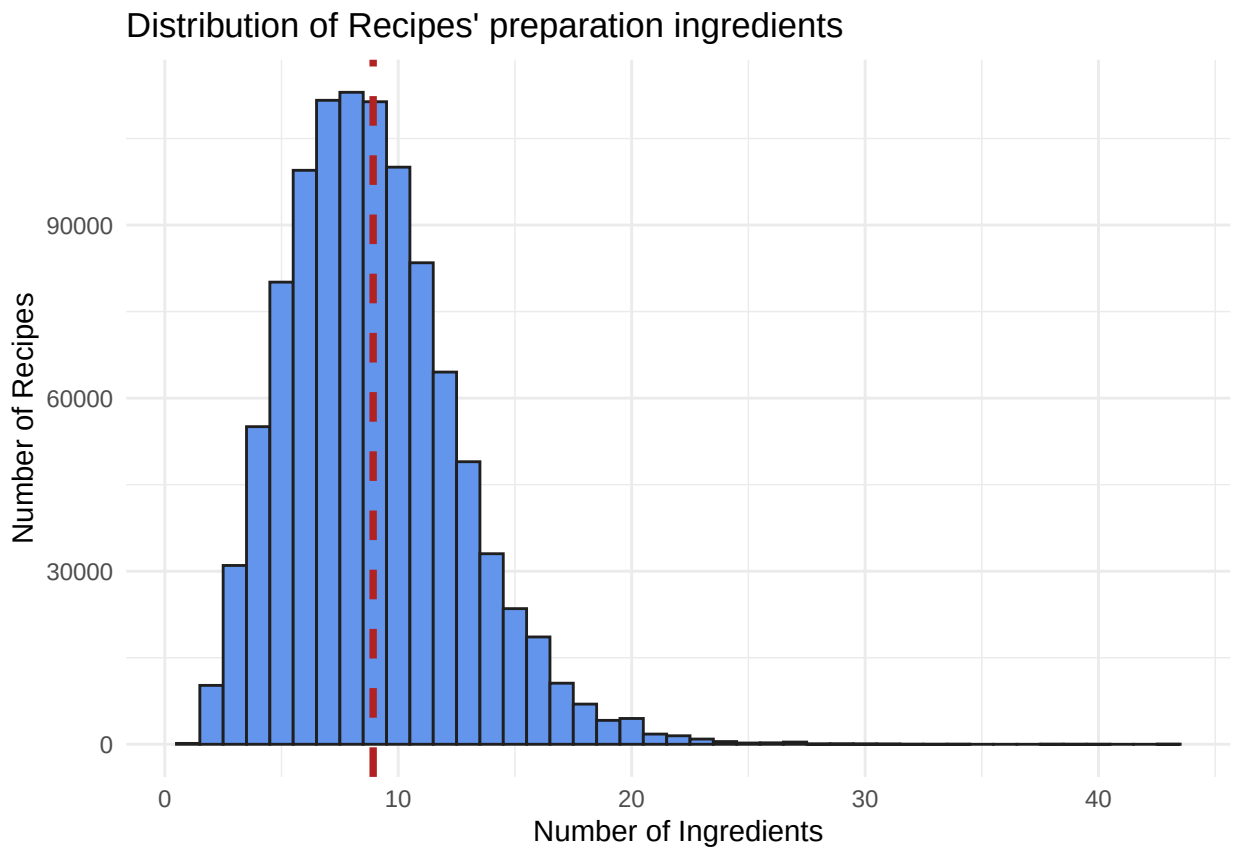
```
# filter out reviews with more than 50 steps of preparation
revRec <- revRec %>%
  filter(n_steps < 50)

revRec %>%
  ggplot()+
  geom_histogram(aes(n_steps),
                 fill = "cornflowerblue", binwidth = 1, color = "grey12") +
  theme_minimal()+
  geom_vline(xintercept = mean(revRec$n_steps),
             color="firebrick", linetype = 2, size = 1.3) +
  ylab("Number of Recipes")+
  xlab("Number of steps")+
  ggtitle("Distribution of Recipes' preparation steps")+
  theme_minimal()
```



Th 'n_ingredients' field stores the last numeric variable natively recorded in the raw dataset. The recorded values ranges from 0 to a maximum of 145 steps with an average of 8.93 ingredients, however the distribution is cut at 50.

```
revRec %>%
  ggplot()+
  geom_histogram(aes(n_ingredients),
                 fill = "cornflowerblue", binwidth = 1, color = "grey12") +
  theme_minimal()+
  geom_vline(xintercept = mean(revRec$n_ingredients),
             color="firebrick", linetype = 2, size = 1.3)+
  ylab("Number of Recipes")+
  xlab("Number of Ingredients")+
  ggtitle("Distribution of Recipes' preparation ingredients")+
  theme_minimal()
```



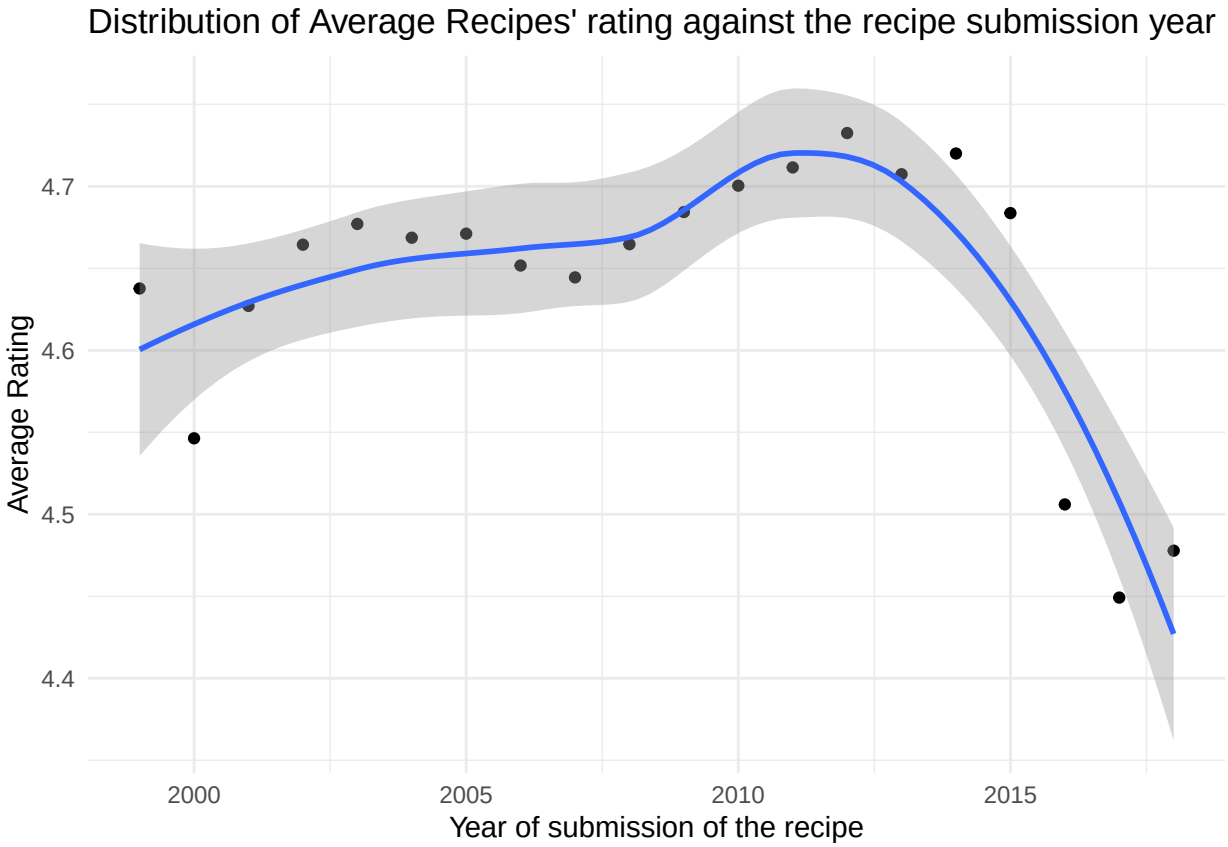
Lastly, two time-dependent relationship could have an effect on the rating distribution, the date of publication of the recipe and the date of publication of the review. This could be visualized as the average rating values against the date values. Both fields show a dome shaped trend with the lowest recorded values at the end and at the beginnings of the time series.

```
### Fix date format
revRec <- revRec %>%
  mutate(submission_month = month(submitted, label = TRUE),
         submission_year = year(submitted),
         review_month = month(date, label = TRUE),
         review_year = year(date))

revRec %>%
  group_by(submission_year) %>%
  summarize(rating = mean(rating)) %>%
  ggplot(aes(submission_year, rating)) +
  geom_point() +
  geom_smooth() +
  theme_minimal() +
  ylab("Average Rating") +
  xlab("Year of submission of the recipe") +
  ggtitle("Distribution of Average Recipes' rating against the recipe submission year")
```

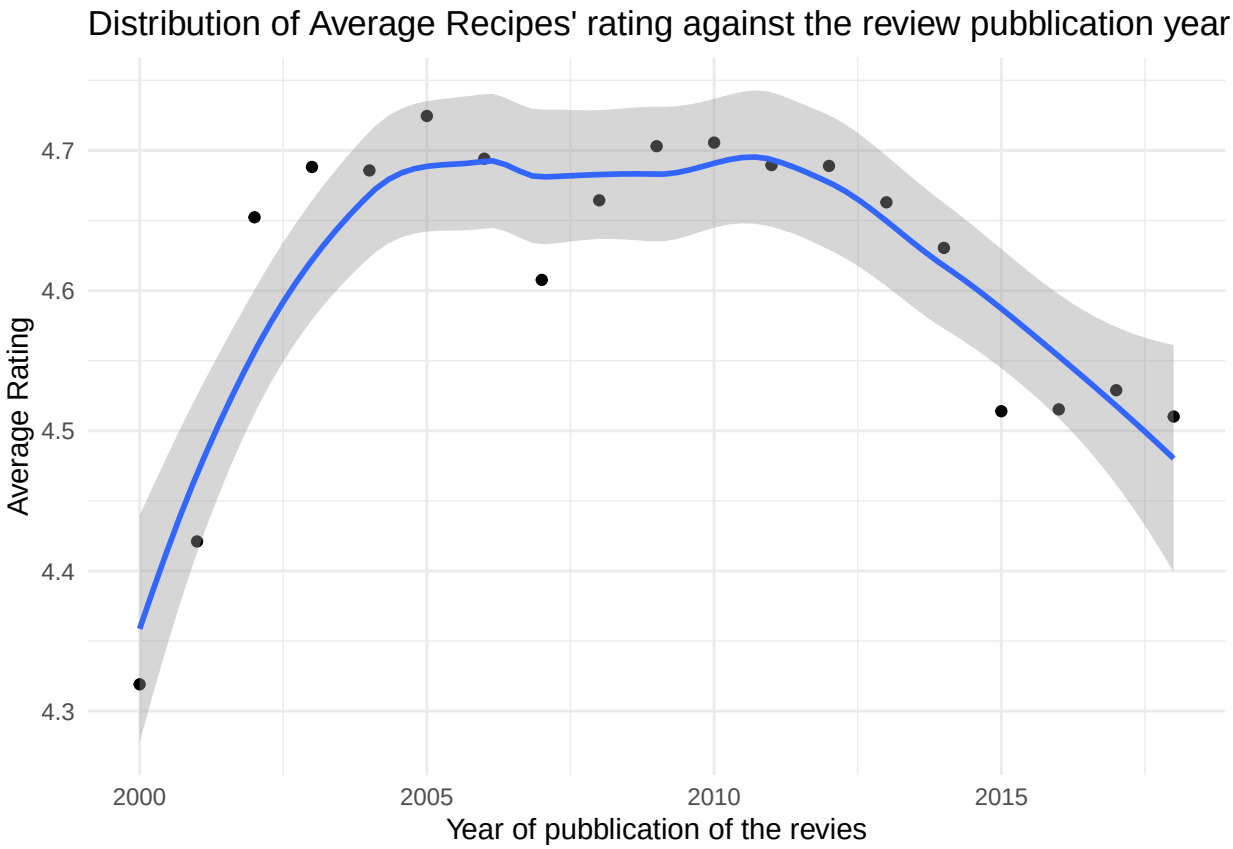
```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```




```
revRec %>%
  group_by(review_year) %>%
  summarize(rating = mean(rating)) %>%
  ggplot(aes(review_year, rating)) +
  geom_point() +
  geom_smooth() +
  theme_minimal() +
  ylab("Average Rating") +
  xlab("Year of pubblication of the revies") +
  ggtitle("Distribution of Average Recipes' rating against the review pubblication year")
```

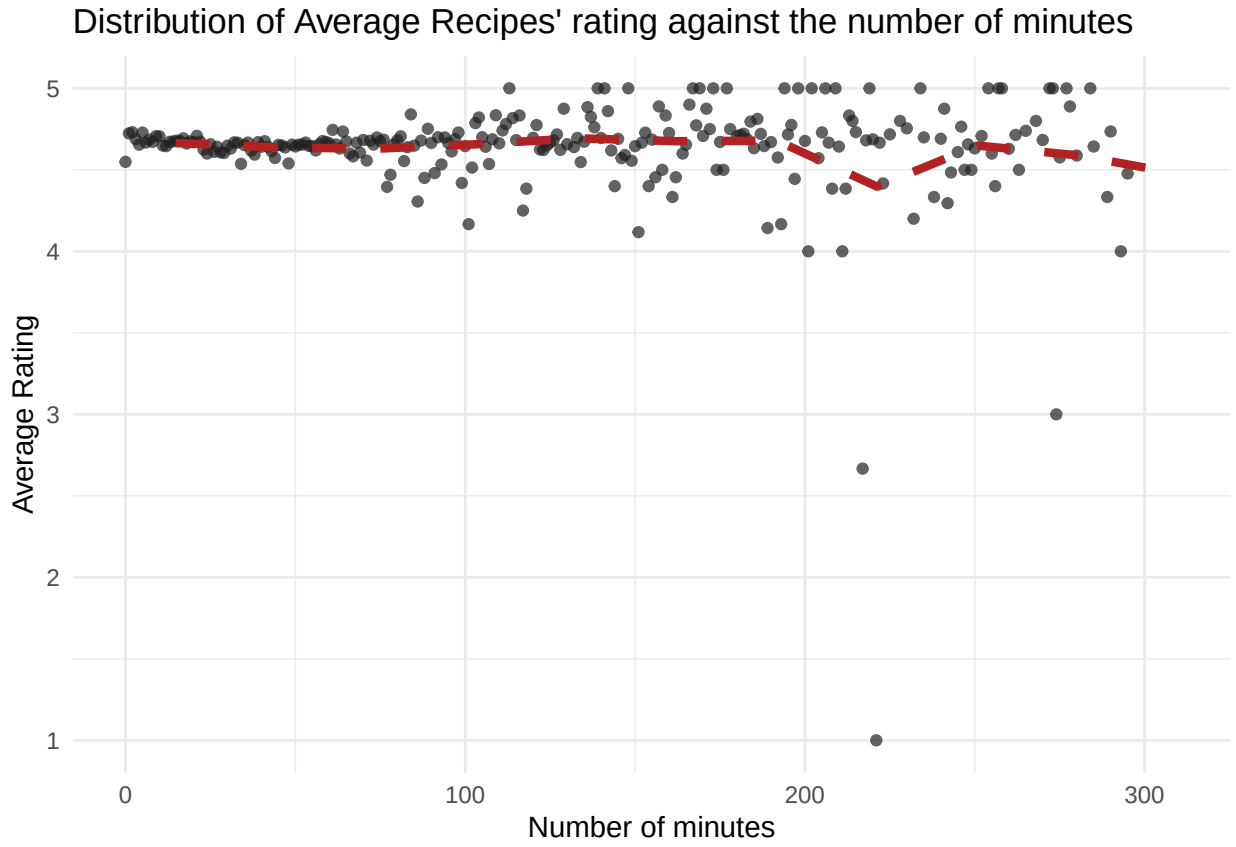
```
## `summarise()` ungrouping output (override with `.groups` argument)
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



After the exploration of the interactions of the ratings with time, let's see if there is some kind of relationship between the ratings and 'minutes', 'n_steps', and 'n_ingredients'.

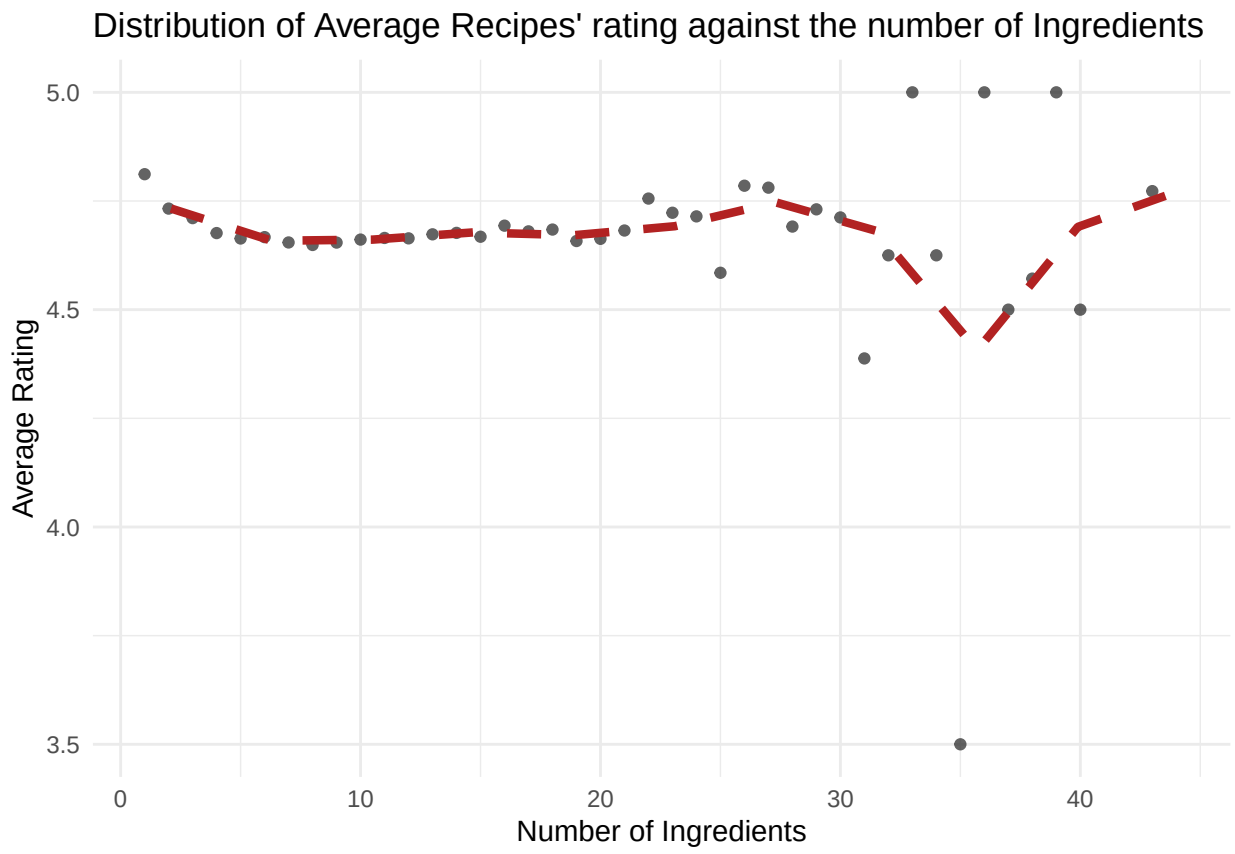
```
revRec %>%
  group_by(minutes) %>%
  summarize(rating = mean(rating)) %>%
  ggplot() +
  geom_point(aes(minutes, rating), alpha = 0.7, color = "grey12") +
  stat_summary_bin(aes(minutes, rating), fun  = mean, bins = 10,
                   color = 'firebrick', geom = 'line', linetype = 2, size = 1.5) +
  theme_minimal() +
  ylab("Average Rating") +
  xlab("Number of minutes") +
  ggtitle("Distribution of Average Recipes' rating against the number of minutes")
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```



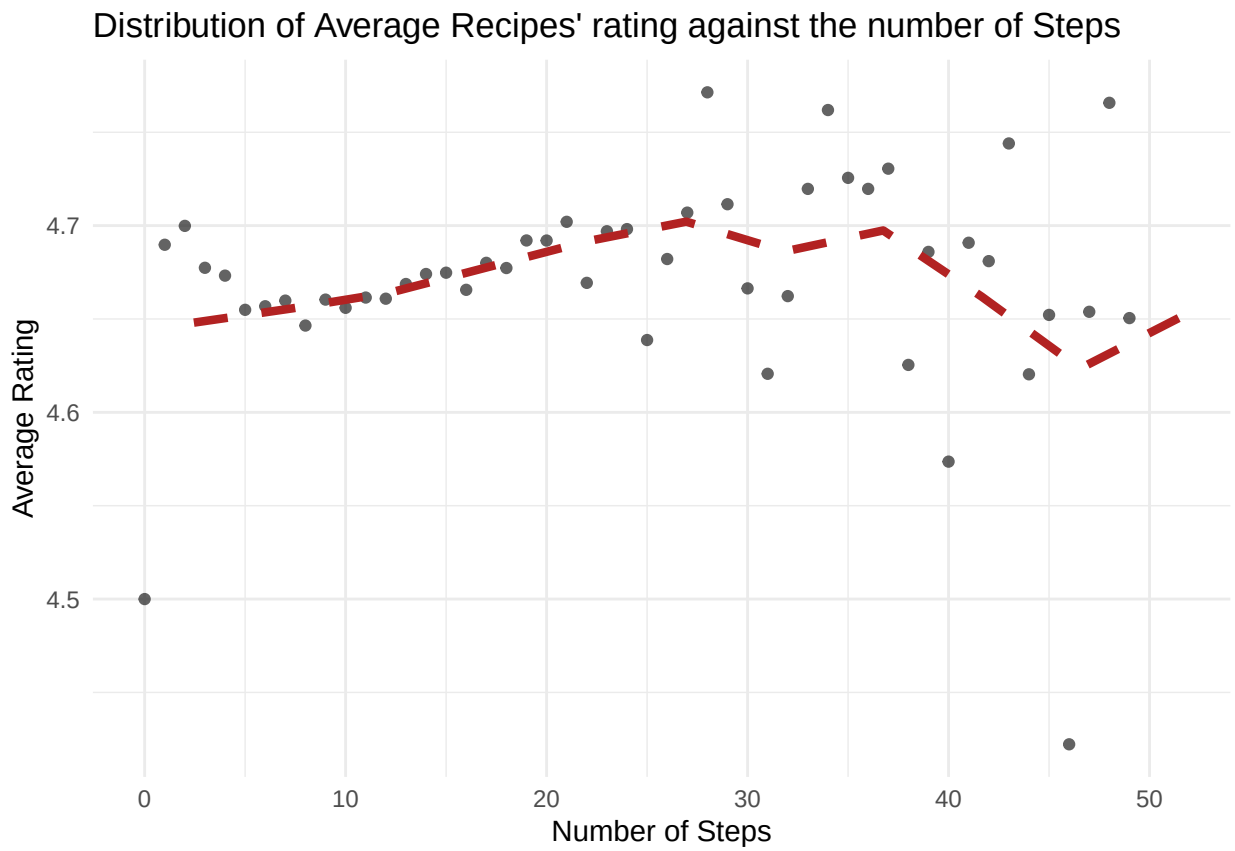

```
revRec %>%
  group_by(n_ingredients) %>%
  summarize(rating = mean(rating)) %>%
  ggplot() +
  geom_point(aes(n_ingredients, rating), alpha = 0.7, color = "grey12") +
  stat_summary_bin(aes(n_ingredients, rating), fun  = mean, bins = 10,
                   color = 'firebrick', geom = 'line', linetype = 2, size = 1.5) +
  theme_minimal() +
  ylab("Average Rating") +
  xlab("Number of Ingredients") +
  ggtitle("Distribution of Average Recipes' rating against the number of Ingredients")
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```




```
revRec %>%
  group_by(n_steps) %>%
  summarize(rating = mean(rating)) %>%
  ggplot() +
  geom_point(aes(n_steps, rating), alpha = 0.7, color = "grey12") +
  stat_summary_bin(aes(n_steps, rating), fun  = mean, bins = 10,
                   color = 'firebrick', geom = 'line', linetype = 2, size = 1.5) +
  theme_minimal() +
  ylab("Average Rating") +
  xlab("Number of Steps") +
  ggtitle("Distribution of Average Recipes' rating against the number of Steps")
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```



Rating modeling

The formatted input is divided into a training and a testing set to measure the performance of the different models.

```
set.seed(1, sample.kind="Rounding") # if using R 3.5 or earlier, use `set.seed(1)`
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
test_index <- createDataPartition(y = revRec$rating, times = 1, p = 0.1, list = FALSE)
train <- revRec[-test_index,]
test <- revRec[test_index,]
```

```
## Warning: The `i` argument of ``[`()` can't be a matrix as of tibble 3.0.0.
## Convert to a vector.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

```
# Check that id and user_id are in both datasets
validation <- test %>%
  semi_join(train, by = "id") %>%
  semi_join(train, by = "user_id")
removed <- anti_join(test, validation)
```

```
## Joining, by = c("name", "id", "minutes", "contributor_id", "submitted", "tags", "nutrition", "n_steps
```

```
train <- rbind(train, removed)
test <- validation
```

The food recipes and interaction dataset can be employed to create a recommendation system analogously to the movielens project. The RMSE is adopted as the error loss function, and as in dsbook 33.7.3 Loss function is defined as: ' $y_{u,i}$ for the rating of recipe i by user u and denote our prediction with $\hat{y}_{u,i}$ and N being the number of user/recipe combinations. The RMSE is mathematically defined as:

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{u,i}\left(\hat{y}_{u,i} - y_{u,i}\right)^2}$$

and implemented in R with the function:

```
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

Incremental complications of the estimator model for the recipe's rating will be tested. The simplest model is represented by the average model, with a single outcome for all the observation as the average of all the ratings in the training set. Mathematically, the random variation can be defined as:

$$Y_{u,i} = \mu + \varepsilon_{u,i}$$

with the independent errors as $\varepsilon_{i,u}$ and μ the observed rating for all movies. Implenting this in R, gives us:

```
# compute the average rating across all the training records
mu_hat <- mean(train$rating)

# compute the rmse between the average and the test set
naive_rmse <- RMSE(test$rating, mu_hat)
rmse_results <- tibble(method = "Just the average", RMSE = naive_rmse)
rmse_results
```

```
## # A tibble: 1 x 2
##   method            RMSE
##   <chr>            <dbl>
## 1 Just the average 0.656
```

Recipe effects

The next step will be the addition of a bias term to take into account that some food's preparation are universally appreciated, or avoided, by a large part of our sample observers. This can be accomplished similarly to the computation of a global average, but this time it will be disaggregated by recipe id to get an average ranking for each recipe. Building from the notation shown for the baseline, we can integrate our model with a new term, b_i as the recipe effect on the final rating outcome:

$$Y_{u,i} = \mu + b_i + \varepsilon_{u,i}$$

In this case, with more than 200000 distinct recipes, it is unpractical to employ a regression model to estimate the b_i bias with the lm function:

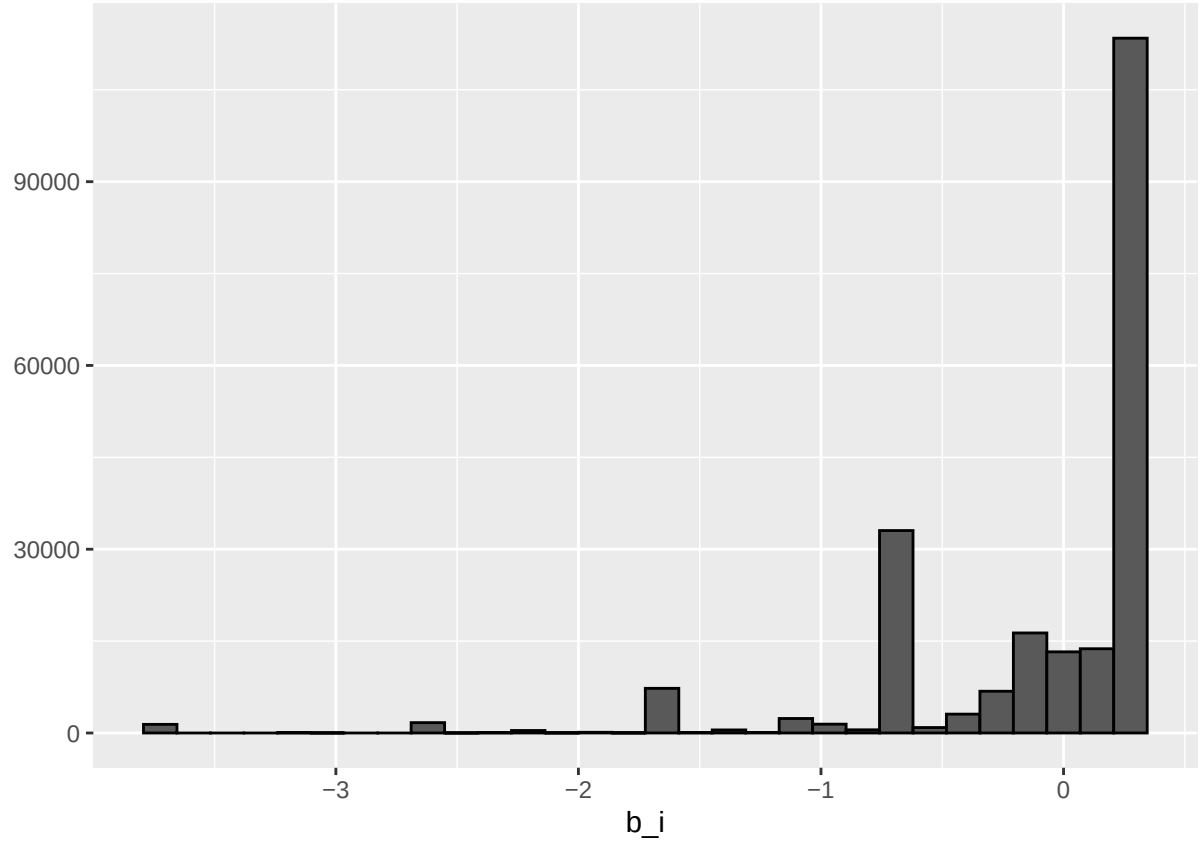
```
fit <- lm(rating ~ as.factor(movieId), data = movielens)
```

Thus the estimation phase is carried out interpreting the least-squares estimate $\hat{b}_i$ as the average of $Y_{u,i} - \hat{\mu}$ for each recipe i:

```
mu_hat <- mean(train$rating)
recipe_avgs <- train %>%
  group_by(id) %>%
  summarize(b_i = mean(rating - mu_hat))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
qplot(b_i, data = recipe_avgs, bins = 30, color = I("black"))
```



The histogram of the residuals of the recipe averages estimation doesn't seem to represent a normal distribution, the majority of ratings lies around zero, it suggests the possibility of either low sample size or absence of this kind of effect. However, once we get the b_i values for each recipe, we can add it to the previously computed global average to get the new predicted ratings taking into account the recipe effect:

```
predicted_ratings <- mu_hat + test %>%
  left_join(recipe_avgs, by='id') %>%
  pull(b_i)

# testNa <- which(is.na(predicted_ratings))
```

```
model_1_rmse <- RMSE(predicted_ratings, test$rating)
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Recipe Effect Model",
                                 RMSE = model_1_rmse))
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	0.6562134
Recipe Effect Model	0.7029821

Unsurprisingly, the model with the recipe effect performs worst than the simple average of all the ratings.

User effects

Similarly to the recipe effect, there should be also a bias made by the tendency of single users to be more or less generous on average compared to the other users. The following model work on the estimation of the average rating for user u defined as:

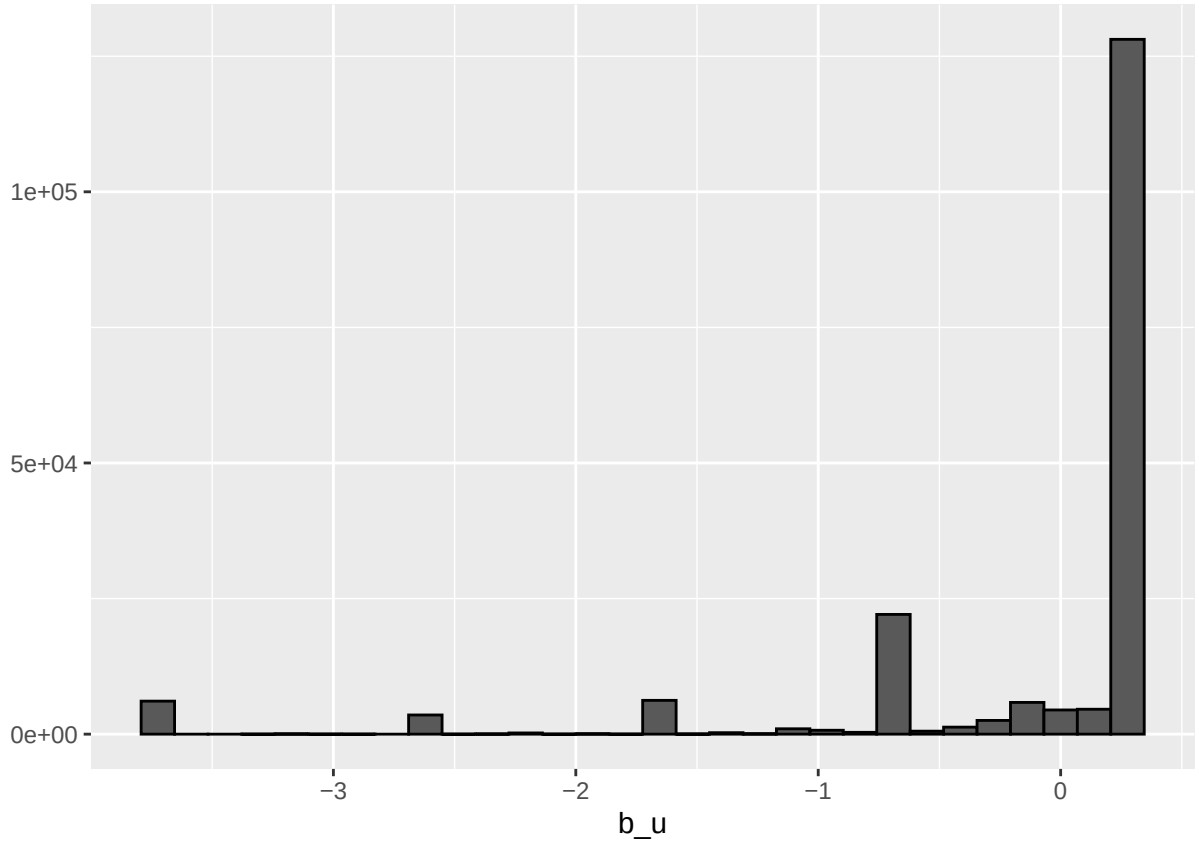
$$Y_{u,i} = \mu + b_u + \varepsilon_{u,i}$$

where b_u is the user effect approximated, as for the recipe effect, with the average of $Y_{u,i} - \hat{\mu}$. The new predictions are computed with:

```
user_avgs <- train %>%
  group_by(user_id) %>%
  summarize(b_u = mean(rating - mu_hat))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
qplot(b_u, data = user_avgs, bins = 30, color = I("black"))
```




```
predicted_ratings <- test %>%
  left_join(user_avgs, by='user_id') %>%
  mutate(pred = mu_hat + b_u) %>%
  pull(pred)

model_2_rmse <- RMSE(predicted_ratings, test$rating)
rmse_results <- bind_rows(rmse_results,
                          tibble(method="User Effects Model",
                                 RMSE = model_2_rmse))
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	0.6562134
Recipe Effect Model	0.7029821
User Effects Model	0.6415721

The RMSE is highly improved than the Recipe Effect alone, but also better than the naive average. Let's see if the integration of the two effects has more predictive power computing the average rating for user u together with the recipe effect into a more complex model. This mathematical definition of the model will now become:

$$Y_{u,i} = \mu + b_i + b_u + \varepsilon_{u,i}$$

where b_u is the user effect. As for the movie effect, we can get a least square estimate of b_u with:

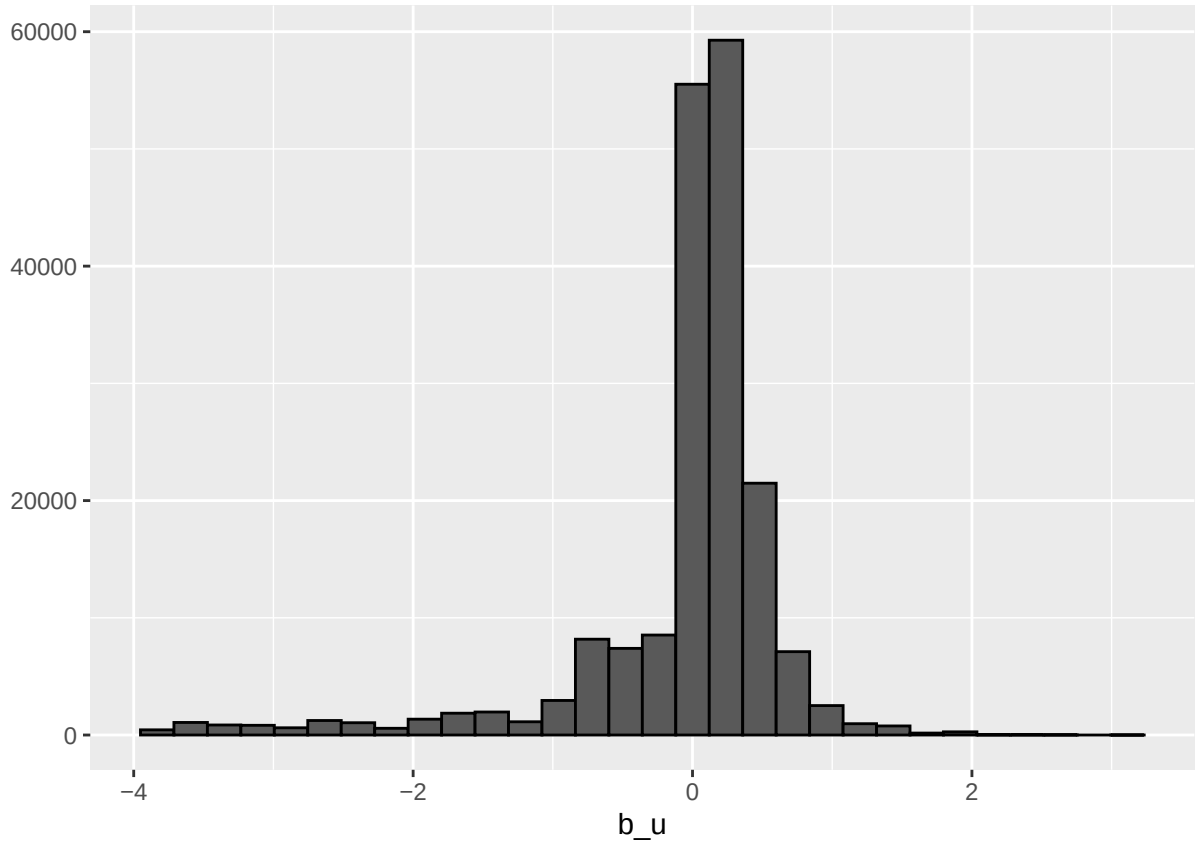
```
lm(rating ~ as.factor(movieId) + as.factor(userId))
```

but, for the same reasons, we will compute the approximation with the average of $y_{u,i} - \hat{\mu} - \hat{b}_i$ and get the new predictions:

```
user_avgs <- train %>%
  left_join(recipe_avgs, by='id') %>%
  group_by(user_id) %>%
  summarize(b_u = mean(rating - mu_hat - b_i))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
qplot(b_u, data = user_avgs, bins = 30, color = I("black"))
```




```
predicted_ratings <- test %>%
  left_join(recipe_avgs, by='id') %>%
  left_join(user_avgs, by='user_id') %>%
  mutate(pred = mu_hat + b_i + b_u) %>%
  pull(pred)
```

The plot of the residual shows a skewed normal distribution. As before, using the RMSE function we will get the new model error:

method	RMSE
Just the average	0.6562134
Recipe Effect Model	0.7029821
User Effects Model	0.6415721
Recipe + User Effects Model	0.6875239

It seems recommended to exclude the recipe effect from subsequent models to avoid to bring more noise than information into the estimation since the inclusion of the recipe effect has a worsening outcome of the model' rmse.

Other effects

The next step is the test of the implementation of the regularization tuning on the user effect:

```
## [1] 4.5
```

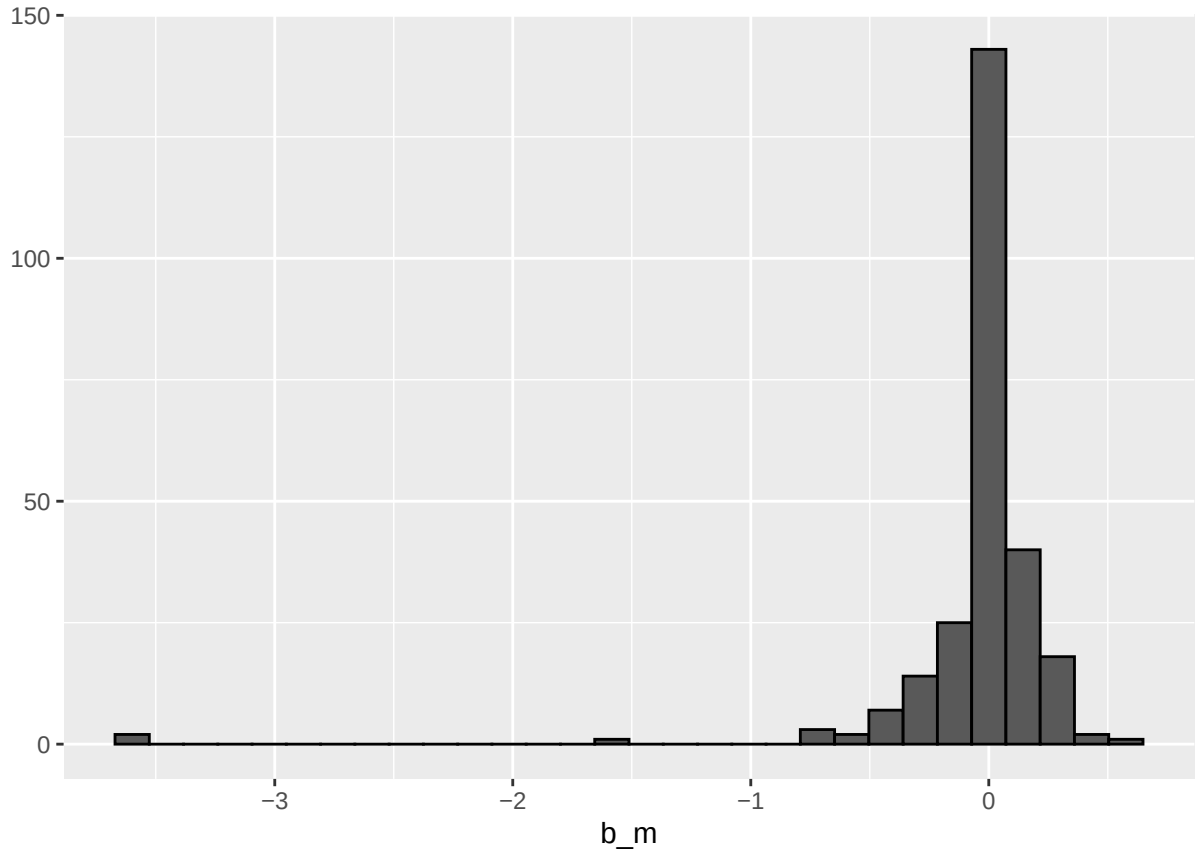
method	RMSE
Just the average	0.6562134
Recipe Effect Model	0.7029821
User Effects Model	0.6415721
Recipe + User Effects Model	0.6875239
Regularized User Effects Model	0.6157418

The regularization of the User effect has lowered the RMSE of the model to 0.616, and now I can try to include other effects. Three other numerical variables can be included in the model: minutes of preparation, number of steps, and the number of ingredients of each recipe. Starting from the average time required for preparing the recipe which is approximated as $y_{u,i} - \hat{\mu} - \hat{b}_u$ with the previously estimated b_u

```
minutes_avgs <- train %>%
  left_join(user_avgs, by='user_id') %>%
  group_by(minutes) %>%
  summarize(b_m = mean(rating - mu_hat - b_u))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
qplot(b_m, data = minutes_avgs, bins = 30, color = I("black"))
```




```
predicted_ratings <- test %>%
  left_join(user_avgs, by='user_id') %>%
  left_join(minutes_avgs, by='minutes') %>%
  mutate(pred = mu_hat + b_u + b_m) %>%
  pull(pred)

model_3_rmse <- RMSE(predicted_ratings, test$rating)
rmse_results <- bind_rows(rmse_results,
                          tibble(method="User + Minutes Effects Model",
                                 RMSE = model_3_rmse))
rmse_results %>% knitr::kable()
```

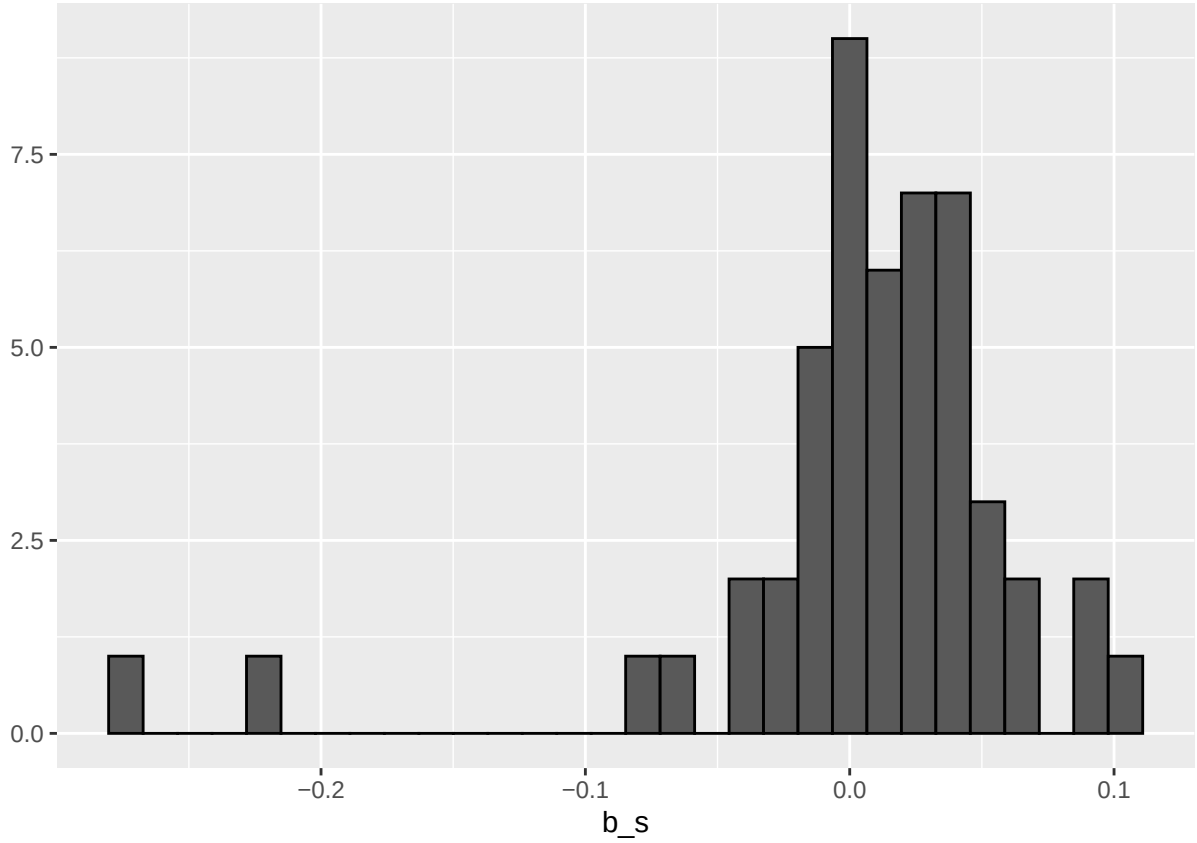
method	RMSE
Just the average	0.6562134
Recipe Effect Model	0.7029821
User Effects Model	0.6415721
Recipe + User Effects Model	0.6875239
Regularized User Effects Model	0.6157418
User + Minutes Effects Model	0.6378467

The prediction is lower than the User only model but higher than the one with the regularization of user effect. The number of steps of each recipe are integrated in a more complex model:

```
step_avgs <- train %>%
  left_join(user_avgs, by='user_id') %>%
  left_join(minutes_avgs, by='minutes') %>%
  group_by(n_steps) %>%
  summarize(b_s = mean(rating - mu_hat - b_u - b_m))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
qplot(b_s, data = step_avgs, bins = 30, color = I("black"))
```




```
predicted_ratings <- test %>%
  left_join(user_avgs, by='user_id') %>%
  left_join(minutes_avgs, by='minutes') %>%
  left_join(step_avgs, by='n_steps') %>%
  mutate(pred = mu_hat + b_u + b_m + b_s) %>%
  pull(pred)

model_4_rmse <- RMSE(predicted_ratings, test$rating)
rmse_results <- bind_rows(rmse_results,
                          tibble(method="User + Minutes + Steps Effects Model",
                                 RMSE = model_4_rmse))
rmse_results %>% knitr::kable()
```

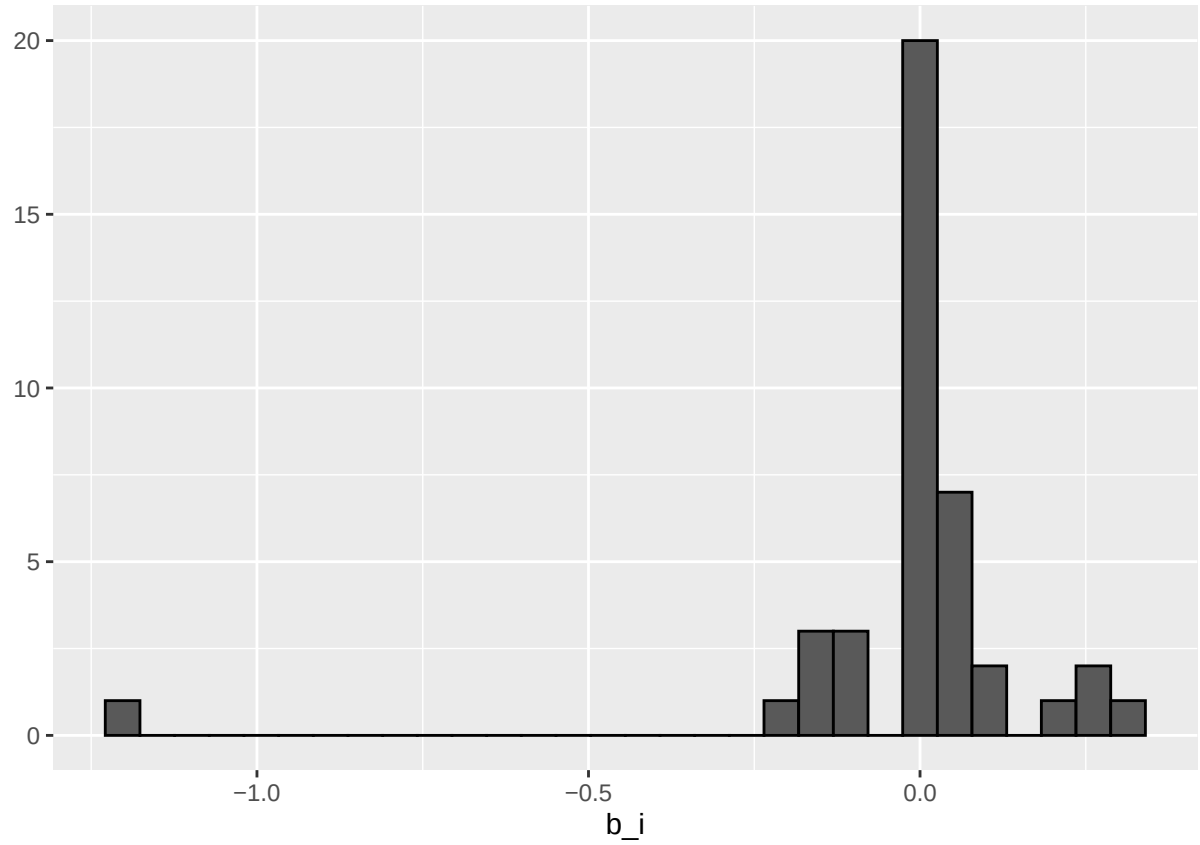
method	RMSE
Just the average	0.6562134
Recipe Effect Model	0.7029821
User Effects Model	0.6415721
Recipe + User Effects Model	0.6875239
Regularized User Effects Model	0.6157418
User + Minutes Effects Model	0.6378467
User + Minutes + Steps Effects Model	0.6376903

The introduction of the 'n_step' effect yields a sligthly lower RMSE. In the next step, the 'n_ingredients' field of the input dataset is added:

```
ingredients_avgs <- train %>%
  left_join(user_avgs, by='user_id') %>%
  left_join(minutes_avgs, by='minutes') %>%
  left_join(step_avgs, by='n_steps') %>%
  group_by(n_ingredients) %>%
  summarize(b_i = mean(rating - mu_hat - b_u - b_m - b_s))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
qplot(b_i, data = ingredients_avgs, bins = 30, color = I("black"))
```



```
predicted_ratings <- test %>%
  left_join(user_avgs, by='user_id') %>%
  left_join(minutes_avgs, by='minutes') %>%
  left_join(step_avgs, by='n_steps') %>%
  left_join(ingredients_avgs, by='n_ingredients') %>%
  mutate(pred = mu_hat + b_u + b_m + b_s + b_i) %>%
  pull(pred)

model_5_rmse <- RMSE(predicted_ratings, test$rating)
rmse_results <- bind_rows(rmse_results,
                          tibble(method="User + Minutes + Steps + Ingredients Effects Model",
                                 RMSE = model_5_rmse))
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	0.6562134
Recipe Effect Model	0.7029821
User Effects Model	0.6415721
Recipe + User Effects Model	0.6875239
Regularized User Effects Model	0.6157418
User + Minutes Effects Model	0.6378467
User + Minutes + Steps Effects Model	0.6376903
User + Minutes + Steps + Ingredients Effects Model	0.6375928

lowering again the RMSE in the order of 10^-4 increments. Lastly, the regularization of the User + Minutes + Steps + Ingredients effect is tested with in the model formally described by $y_{u,i} - \hat{\mu} - \hat{b}_u - \hat{b}_m - \hat{b}_s - \hat{b}_i$ with b_u as the user effect, b_m the minutes of preparations, b_s as the number of steps, and b_i as the number of steps effect:

```
## [1] 4.25
```

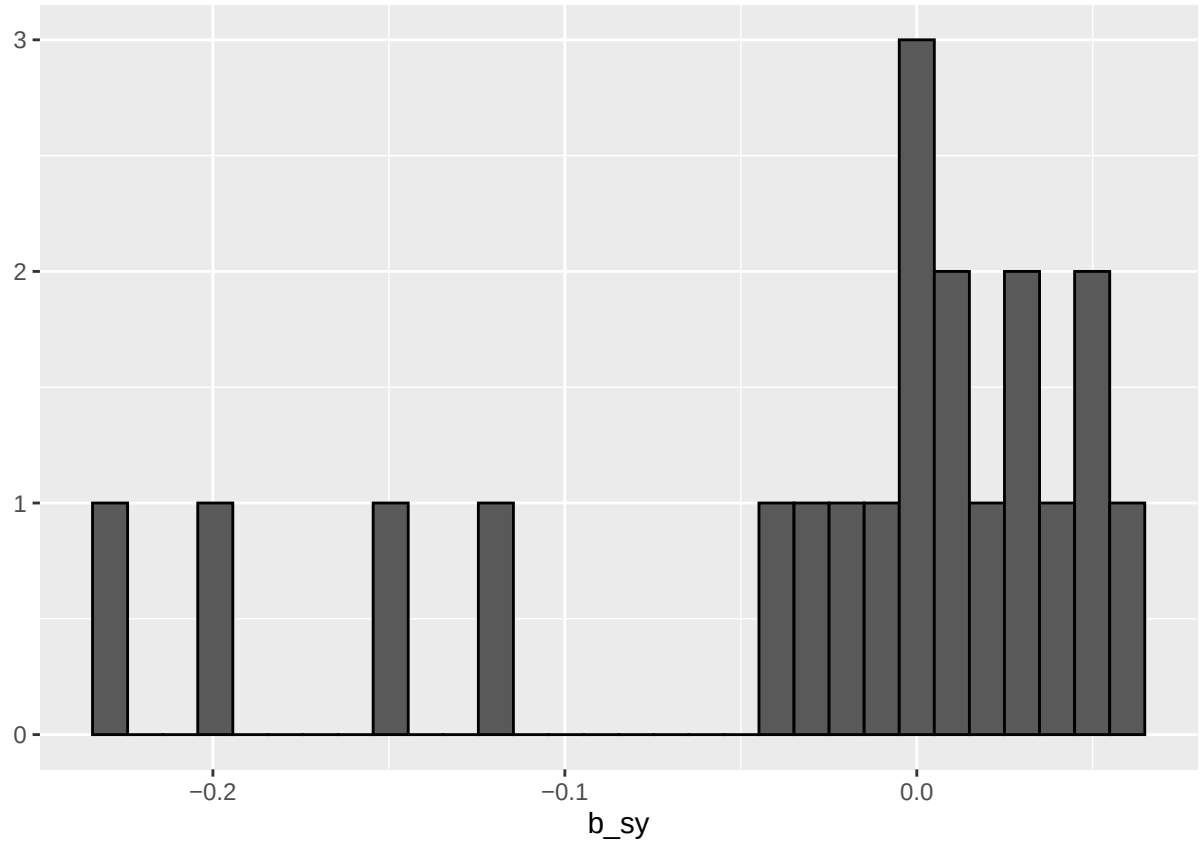
method	RMSE
Just the average	0.6562134
Recipe Effect Model	0.7029821
User Effects Model	0.6415721
Recipe + User Effects Model	0.6875239
Regularized User Effects Model	0.6157418
User + Minutes Effects Model	0.6378467
User + Minutes + Steps Effects Model	0.6376903
User + Minutes + Steps + Ingredients Effects Model	0.6375928
Regularized User + Minutes + Steps + Ingredients Effects Model	0.6158355

The regularization procedure takes the model' RMSE to circa 0.6 but it is comparable to the regularized model with only the user effect. furthermore, it could be interesting to explore the possible time-relationship between the year of submission and/or review of a recipe to improve the RMSE of the model. First, the year of submission is include as the approximated average from $y_{u,i} - \hat{\mu} - \hat{b}_{sy}$:

```
subY_avgs <- train %>%
  group_by(submission_year) %>%
  summarize(b_sy = mean(rating - mu_hat))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
qplot(b_sy, data = subY_avgs, bins = 30, color = I("black"))
```




```
predicted_ratings <- test %>%
  left_join(subY_avgs, by='submission_year') %>%
  mutate(pred = mu_hat + b_sy) %>%
  pull(pred)

model_2_rmse <- RMSE(predicted_ratings, test$rating)
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Submission Year Effects Model",
                                 RMSE = model_2_rmse))
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	0.6562134
Recipe Effect Model	0.7029821
User Effects Model	0.6415721

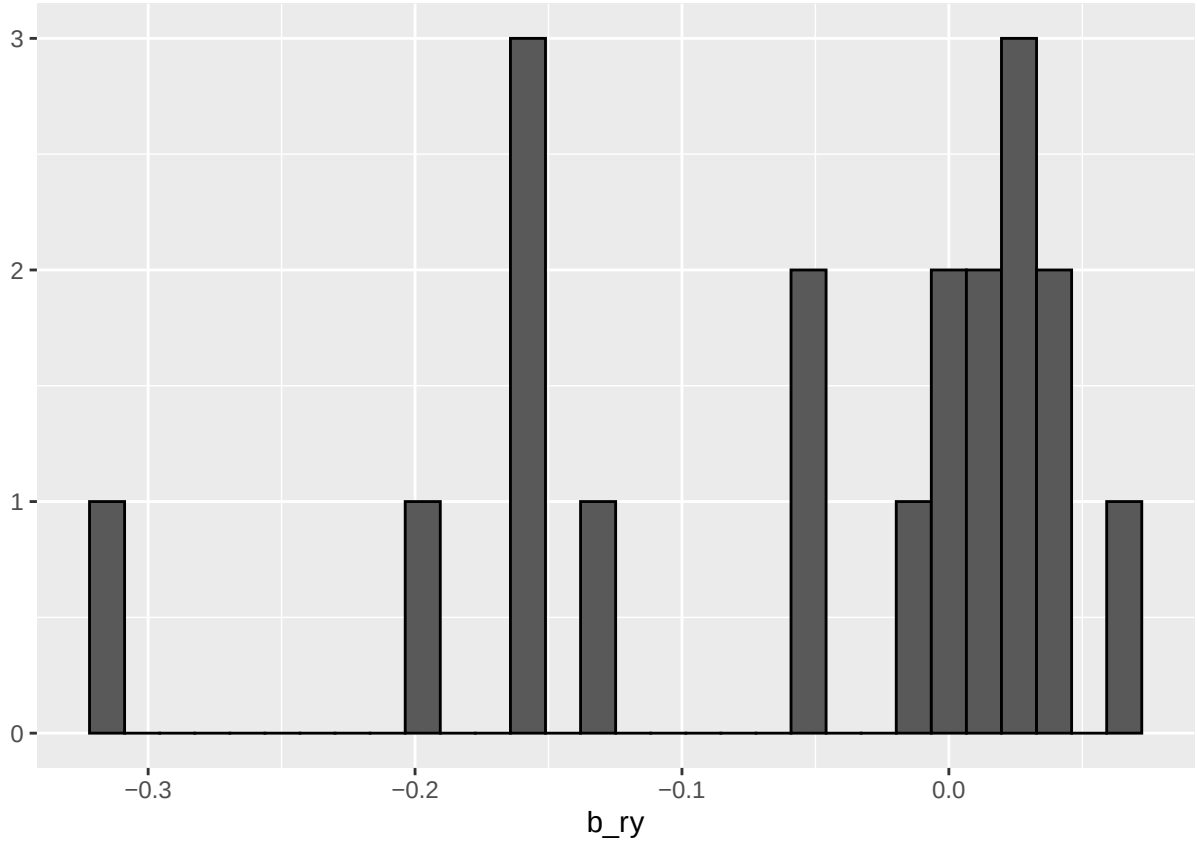
method	RMSE
Recipe + User Effects Model	0.6875239
Regularized User Effects Model	0.6157418
User + Minutes Effects Model	0.6378467
User + Minutes + Steps Effects Model	0.6376903
User + Minutes + Steps + Ingredients Effects Model	0.6375928
Regularized User + Minutes + Steps + Ingredients Effects Model	0.6158355
Submission Year Effects Model	0.6558725

the RMSE of this model is again quite similar to the outcome of the naive model with just the average. Testing the effect of the year of the review registration, the result is the same with a final RMSE of about 0.65.

```
revY_avgs <- train %>%
  left_join(subY_avgs, by='submission_year') %>%
  group_by(review_year) %>%
  summarize(b_ry = mean(rating - mu_hat - b_sy))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
qplot(b_ry, data = revY_avgs, bins = 30, color = I("black"))
```



```
predicted_ratings <- test %>%
  left_join(subY_avgs, by='submission_year') %>%
  left_join(revY_avgs, by='review_year') %>%
  mutate(pred = mu_hat + b_sy + b_ry) %>%
  pull(pred)


model_2_rmse <- RMSE(predicted_ratings, test$rating)
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Review Year Effects Model",
                                 RMSE = model_2_rmse))
rmse_results %>% knitr::kable()
```

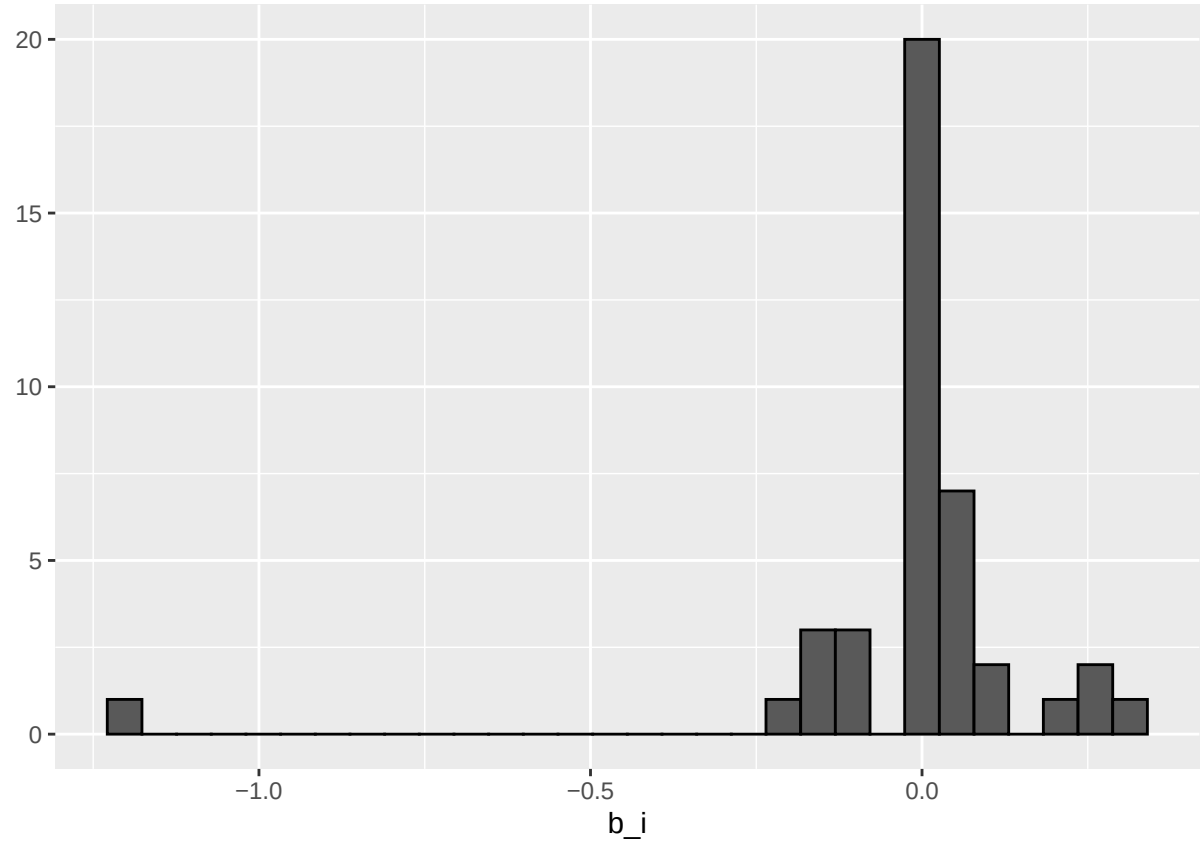
method	RMSE
Just the average	0.6562134
Recipe Effect Model	0.7029821
User Effects Model	0.6415721
Recipe + User Effects Model	0.6875239
Regularized User Effects Model	0.6157418
User + Minutes Effects Model	0.6378467
User + Minutes + Steps Effects Model	0.6376903
User + Minutes + Steps + Ingredients Effects Model	0.6375928
Regularized User + Minutes + Steps + Ingredients Effects Model	0.6158355
Submission Year Effects Model	0.6558725
Review Year Effects Model	0.6551241

Also, the plot of the residuals of the approximation of the submission and review year terms, highligths the scarcity of reference points iwthin the time series. Lastly, the model including all the variable presented in this analysis is estimated from the averages of $y_{u,i} - \hat{\mu} - \hat{b}_u - \hat{b}_m - \hat{b}_s - \hat{b}_{sy} - \hat{b}_{ry}$:

```
revY_avgs <- train %>%
  left_join(user_avgs, by='user_id') %>%
  left_join(minutes_avgs, by='minutes') %>%
  left_join(step_avgs, by='n_steps') %>%
  left_join(ingredients_avgs, by='n_ingredients') %>%
  left_join(subY_avgs, by='submission_year') %>%
  group_by(review_year) %>%
  summarize(b_ry = mean(rating - mu_hat - b_u - b_m - b_s - b_i - b_sy))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
qplot(b_i, data = ingredients_avgs, bins = 30, color = I("black"))
```




```
predicted_ratings <- test %>%
  left_join(user_avgs, by='user_id') %>%
  left_join(minutes_avgs, by='minutes') %>%
  left_join(step_avgs, by='n_steps') %>%
  left_join(ingredients_avgs, by='n_ingredients') %>%
  left_join(subY_avgs, by='submission_year') %>%
  left_join(revY_avgs, by='review_year') %>%
  mutate(pred = mu_hat + b_u + b_m + b_s + b_i + b_sy + b_ry) %>%
  pull(pred)

model_5_rmse <- RMSE(predicted_ratings, test$rating)
rmse_results <- bind_rows(rmse_results,
                          tibble(method="User + Minutes + Steps + Ingredients + Sub + Rev Effects Model
                                 RMSE = model_5_rmse))
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	0.6562134
Recipe Effect Model	0.7029821
User Effects Model	0.6415721
Recipe + User Effects Model	0.6875239
Regularized User Effects Model	0.6157418
User + Minutes Effects Model	0.6378467
User + Minutes + Steps Effects Model	0.6376903
User + Minutes + Steps + Ingredients Effects Model	0.6375928

method	RMSE
Regularized User + Minutes + Steps + Ingredients Effects Model	0.6158355
Submission Year Effects Model	0.6558725
Review Year Effects Model	0.6551241
User + Minutes + Steps + Ingredients + Sub + Rev Effects Model	0.6367551

and regularized with:

```
## [1] 4.5
```

method	RMSE
Just the average	0.6562134
Recipe Effect Model	0.7029821
User Effects Model	0.6415721
Recipe + User Effects Model	0.6875239
Regularized User Effects Model	0.6157418
User + Minutes Effects Model	0.6378467
User + Minutes + Steps Effects Model	0.6376903
User + Minutes + Steps + Ingredients Effects Model	0.6375928
Regularized User + Minutes + Steps + Ingredients Effects Model	0.6158355
Submission Year Effects Model	0.6558725
Review Year Effects Model	0.6551241
User + Minutes + Steps + Ingredients + Sub + Rev Effects Model	0.6367551
Regularized User + Minutes + Steps + Ingredients + Years Effects Model	0.6154559

There is a small improvement in the RMSE but it performs just like the simpler model with only the regularized user effect and a 0.015 worst than the one with Regularized User + Minute + Steps + Ingredients.

Discussion

This project is focused on the exploratory data analysis and tentative modelling of a dataset of food recipes and their reviews. The raw data, consisting of more than 1 million of reviews and 200000 recipes, had the characteristics to be a 'large dataset' and a potential candidate for a recommendation system similar to the movielens project. However, the exploratory analyses conducted has highlighted an unexpected difficulty in obtaining a well-performing model. The naive model of 'Just the average' recorded an RMSE of 0.6562134 that represented a solid baseline that couldn't be improved easily. In hindsight, the high scoring of the naive model, together with the plot of the distribution of Recipe's Ratings were two important indications of low estimation performance.

Even from the visualization of the distribution of the scores from one to five, it was evident that there is an oversaturation of maximum votes (5) that make it more difficult to reconstruct a trend with a non-normal distribution pushed to the right. Moreover, also from the scatter graphs of the variables 'minutes', 'n_steps', and 'n_ingredients' compared to the mean value of the rating, show a very narrow range within which the values can vary that in practice is concentrated between 4 and 5.

Besides, although not reported in this paper, several other machine learning approaches have been attempted to try to predict the rating outcome through regression or classification techniques and I briefly report some of the considerations that can be drawn from these. Regarding classification, as stated earlier, it is not possible to use the recipe id or review id to estimate a linear regression model using these two variables as factors (it would require about 400 Gb of memory) and they were therefore excluded from the procedure. Regression

models with lm, or with generalized linear models, on the other variables still reported values similar to or worse than those from the approximation described above. On the other hand, for the classification systems, the observed rating was converted to factor, but none of the models tested (glm and cart) were able to produce an acceptable result. Both showed a tendency to always predict the most abundant class (rating 5) and sometimes the 4th.

Overall, this study has allowed understanding how conceptually similar case studies, are profoundly different in terms of the complex interactions that are attempted to be modelled and the more complex mathematical models are not always a guarantee of success in prediction of results. In particular, given the specific application of recommendation system discussed here, as far as the data collection part is concerned, I would suggest trying to favour more 'intermediate' votes to have more information in the analysis phase. As for the data analysis, if more time was available, there would be many other sources of information to analyze. For example, the 'nutrition' field contains an array with the amounts of protein, fat, sugar and fibre in each recipe. This information lends itself very well to a clustering analysis of the recipes both for mere knowledge discovery and eventual categorization of the main types of food and for the subsequent use of these clusters as subgroups for the regression exercises of the rating concerning the other variables. Similarly, the 'ingredients' field is also a list of strings representing the different ingredients that make up a recipe and could be an interesting target for research between 'nutrition' characteristics and ingredient types. Also, there are two descriptive fields, text-only, for recipe description or review description that both can lend themselves very well to text data mining exercises for both regression model tuning and sentiment analysis applications.

Bibliography

https://rafalab.github.io/dsbook/

https://www.kaggle.com/shuyangli94/food-com-recipes-and-user-interactions?select=RAW_recipes.csv