Prediction of wine Recommendation.

```
#************************************************************
# Author: D. Kassa
# Date: June 12, 2020
#************************************************************
#  Version  1.00
#
#************************************************************
```

```
#*****************************************************************************
#
# Overview
#
#*****************************************************************************
```

The goal of this project is to utilize the lesson learned in HarvardX education; and to demonstrate understanding in R programming and basic machine learning. This report will determine if different machine learning models, and the effect of country where the wine was produced, and the sell price of wine will affect rating prediction.

```
#*****************************************************************************
#
# Introduction
#
#*****************************************************************************
```

Wine is one of the oldest alcohol beverages that have been consumed by humans for the last several thousand years. The earliest evidence of wine being used as alcoholic beverage was discovered in China around 7000 BC; Georgia from 6000 BC, Iran from 5000 BC, and Italy from 4000 BC.

Wine rating has point scale classification known as a 100-point-based system. The 95-100 score reflects an accord that the bottle is of excellent quality; whereas low scores implies the wine is of lesser quality and eventually scaled down to 50-74, not recommended.

The assessment of wine review points are as follows:

```
95-100 Classic: a great wine.
90-94 Outstanding: a wine of superior character and style.
85-89 Very good: a wine with special qualities.
80-84 Good: a solid, well-made wine.
75-79 Mediocre: a drinkable wine that may have minor flaws.
50-74 Not recommended.
```

The idea behind wine rating scale is to normalize wine standard. Different wine reviewers have different unique palate they rely on, as well as fondness towards desiring particular wine. Wine ratings value is more of reference point for sale price and quality of wine. However, the review point can be a perception to gauging price, rather the actual quality of wine by consumers.

Consumers could benefit from the numbers-based approach, since it will give a shared understanding behind the classification of wine. As a result, consumers have tendency to reflect the higher the rating scale, the more expensive wine could cost. In reality, wine ratings are subjective, but the review points can be helpful guide to use in order to satisfy personal appetite for wine drinkers. This report will demonstrate if wine rating prediction will improve by country the wine was produced, and selling price.

```
#*******************************************************************************
#
# Dataset
#
#*******************************************************************************
```

Dataset was downloaded from Kaggle: https://www.kaggle.com/zynicide/wine-reviews; and it contains around 120K rows. After cleaning the data, the metadata of interest are:

wineId: Bottle Unique Identifier
country: Country of grape
points: Rating of review
price: Cost of bottle wine
province: State or Province

In order to develop recommendation system, column rating and countryId was added to the downloaded dataframe. Rating column was populated with values from 1 to 5. The rating column scale is directly conforming to the points column listed below:

```
#*******************************************************************************
#  Ranking the review points to a scale of 1 - 5 rating
#
#  Rating     Points  Description
#  ------     ------  ---------------
#  5 -      95-100 Classic: a great wine.
#  4 -      90-94 Outstanding: a wine of superior character and style.
#  3 -      85-89 Very good: a wine with special qualities.
#  2 -      80-84 Good: a solid, well-made wine.
#  1 -      75-79 Mediocre: a drinkable wine that may have minor flaws.
#  0 -      50-74 Not recommended.
#*******************************************************************************
```

CountryId column reflects numerical representation of a country name; and data analysis was performed to understand the dataset; and gain more insight.

To develop the machine learning model, 10% of the original rows of data was extracted for validation sample; and the remaining 90% was split again into 90% for training, 10% testing.

```
#*******************************************************************************
#
# Executive Summary
#
#*******************************************************************************
```

The emphasis on this project is to use different models and improve wine rating by using country of wine produced and bottle of wine price effect. We will use these effects to derive the Root Mean Square Error (RMSE) value, and compare results to different machine learning models.

RMSE is the standard deviation of the prediction errors, a measure of how far from the regression line data points will be; and it is defined as:

$$RMSE = \sqrt{\frac{1}{N}\sum_{p,c}(\hat{y}_{p,c} - y_{p,c})^2}$$

N is the number of country-price combinations, yp,c is the ratings for country c by price p, and y^p,c is our prediction.

Two different models are used in this project. The first model is linear regression, and the second model is matrix factorization. The Matrix Factorization model is from the recosystem package. Recosystem is an R wrapper of the LIBMF library, an open source library for A Library for Parallel Matrix Factorization in Shared-memory Systems.

After comparing the models the lowest RMSE value was achieved by the regularized linear regression model, and a value of 0.5570853 was obtained.

```
#*****************************************************************************
#
# Methods and Analysis
#*****************************************************************************
```

For simplicity purpose, this project will not segregate data into different wine type (red, white, rose, etc).

Data downloaded from Kaggle website, must be available in the working directory to run the Rmd and R files.

The script begins by determining locally installed R packages to run the recommendation script. For packages that are not available in the current local RStudio session, the script will automatically download and install missing packages.

After performing preliminary analysis; temporary tables will be created for plotting different graphs, and to gain additional insight to the dataset.

Before training the models, the data is split into training, testing, and validation sample.

The initial model regulates base line (assumes no effect by other parameters). We will enhance the model by incorporating country effect; followed by country + price effect to reduce RMSE value. We will further improve the model by integrating tuning parameter to the country + price effect. The minimum lambda value derived will be used to achieve lower RMSE value.

Matrix Factorization (MF) was considered as the second model. The MF is part of the recosystem package. After training, testing and evaluating the MF model, all the RMSE values were tabulated to determine better recommendation system.

Ultimately, the machine learning algorithm that achieves the lowest RMSE value, will be the better recommending system.

```
#*****************************************************************************
#Install and Load Required packages
#*****************************************************************************
```

```
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(dplyr)) install.packages("dplyr", repos = "http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(data.table)) install.packages("data.table", repos = "http://cran.us.r-project.org")
if(!require(dslabs)) install.packages("dslabs", repos = "http://cran.us.r-project.org")
```

```r
if(!require(stringr)) install.packages("stringr", repos = "http://cran.us.r-project.org")
if(!require(knitr)) install.packages("knitr", repos = "http://cran.us.r-project.org")
if(!require(tinytex)) install.packages("tinytex", repos = "http://cran.us.r-project.org")
if(!require(ggthemes)) install.packages("ggthemes", repos = "http://cran.us.r-project.org")
if(!require(ggrepel)) install.packages("ggrepel", repos = "http://cran.us.r-project.org")
if(!require(gridExtra)) install.packages("gridExtra", repos = "http://cran.us.r-project.org")
if(!require(ggfortify)) install.packages("ggfortify", repos = "http://cran.us.r-project.org")
if(!require(ggplot2)) install.packages("ggplot2", repos = "http://cran.us.r-project.org")
if(!require(lubridate)) install.packages("lubridate", repos = "http://cran.us.r-project.org")
if(!require(tidyr)) install.packages("tidyr", repos = "http://cran.us.r-project.org")
if(!require(e1071)) install.packages("e1071", repos = "http://cran.us.r-project.org")
if(!require(recosystem)) install.packages("recosystem", repos = "http://cran.us.r-project.org")
```

```r
library(tidyverse)
library(dplyr)
library(caret)
library("data.table")
library(dslabs)
library(stringr)
library(knitr)
library(tinytex)
library(ggthemes)
library(ggrepel)
library(gridExtra)
library(ggfortify)
library(ggplot2)
library(lubridate)
library(tidyr)
library(e1071)
library(recosystem)
```

```r
#*****************************************************************************
# Load downloaded data into wine_kaggle dataframe, and examine dataset.
#*****************************************************************************
```

```r
wine_kaggle <- read_csv(unzip("wine_review_data.csv.zip"))

names(wine_kaggle)
```

```
##  [1] "wineId"                "country"               "description"
##  [4] "designation"           "points"                "price"
##  [7] "province"              "region_1"              "region_2"
## [10] "taster_name"           "taster_twitter_handle" "title"
## [13] "variety"               "winery"
```

```r
str(wine_kaggle)
```

```
## tibble [129,971 x 14] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ wineId               : num [1:129971] 1 2 3 4 5 6 7 8 9 10 ...
##  $ country              : chr [1:129971] "Portugal" "US" "US" "US" ...
##  $ description          : chr [1:129971] "This is ripe and fruity, a wine that is smooth while still
```

```
##  $ designation          : chr [1:129971] "Avidagos" NA "Reserve Late Harvest" "Vintner's Reserve Wil
##  $ points               : num [1:129971] 87 87 87 87 87 87 87 87 87 87 ...
##  $ price                : num [1:129971] 15 14 13 65 15 16 24 12 27 19 ...
##  $ province             : chr [1:129971] "Douro" "Oregon" "Michigan" "Oregon" ...
##  $ region_1             : chr [1:129971] NA "Willamette Valley" "Lake Michigan Shore" "Willamette Va
##  $ region_2             : chr [1:129971] NA "Willamette Valley" NA "Willamette Valley" ...
##  $ taster_name          : chr [1:129971] "Roger Voss" "Paul Gregutt" "Alexander Peartree" "Paul Greg
##  $ taster_twitter_handle: chr [1:129971] "@vossroger" "@paulgwine " NA "@paulgwine " ...
##  $ title                : chr [1:129971] "Quinta dos Avidagos 2011 Avidagos Red (Douro)" "Rainstorm 
##  $ variety              : chr [1:129971] "Portuguese Red" "Pinot Gris" "Riesling" "Pinot Noir" ...
##  $ winery               : chr [1:129971] "Quinta dos Avidagos" "Rainstorm" "St. Julian" "Sweet Cheek
##  - attr(*, "spec")=
##   .. cols(
##   ..   wineId = col_double(),
##   ..   country = col_character(),
##   ..   description = col_character(),
##   ..   designation = col_character(),
##   ..   points = col_double(),
##   ..   price = col_double(),
##   ..   province = col_character(),
##   ..   region_1 = col_character(),
##   ..   region_2 = col_character(),
##   ..   taster_name = col_character(),
##   ..   taster_twitter_handle = col_character(),
##   ..   title = col_character(),
##   ..   variety = col_character(),
##   ..   winery = col_character()
##   .. )
```

```
#*******************************************************************************
# Create dataframe wine_data by selecting columns wineId, country, points,
# and price columns; and determine row count of the dataframe.
#*******************************************************************************

wine_data <- subset(wine_kaggle, select = c(wineId, country, points, price))
nrow(wine_data)
```

```
## [1] 129971
```

```
#*******************************************************************************
# Exclude rows that do not contain values, and thar are duplicate from the dataset
#*******************************************************************************
wine_review<- wine_data%>%drop_na()%>%distinct()
```

```
#*******************************************************************************
# Determine observations and variables of the cleaned dataframe (wine_review)
#*******************************************************************************
dim(wine_review)
```

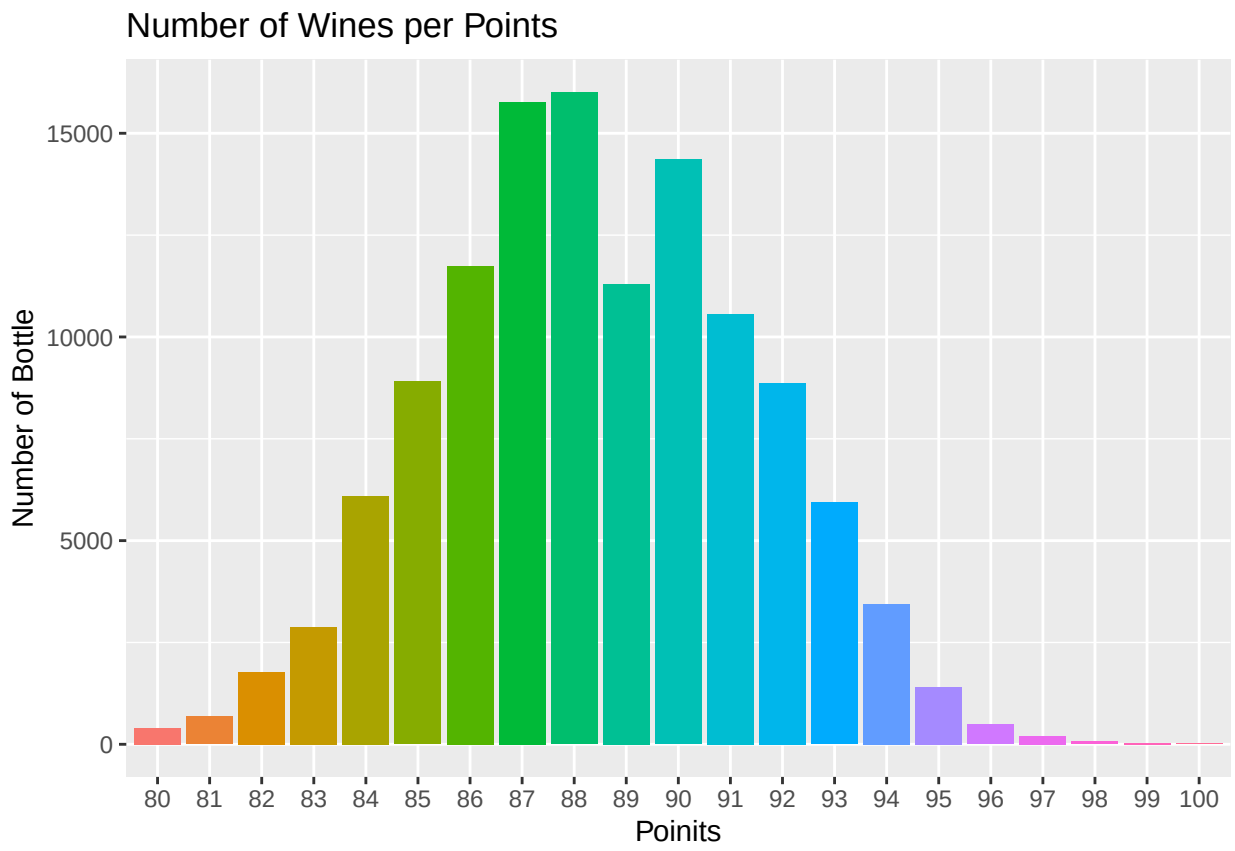
```
## [1] 120916      4
```

```r
#*********************************************************************************
# Determine number of row counts per points
#*********************************************************************************
wine_review%>%
group_by(points) %>%
summarize(n = n())%>%
print(n=21)
```

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

```
## # A tibble: 21 x 2
##    points     n
##     <dbl> <int>
##  1     80   395
##  2     81   680
##  3     82  1772
##  4     83  2886
##  5     84  6097
##  6     85  8901
##  7     86 11740
##  8     87 15761
##  9     88 16005
## 10     89 11306
## 11     90 14354
## 12     91 10559
## 13     92  8865
## 14     93  5935
## 15     94  3449
## 16     95  1406
## 17     96   482
## 18     97   207
## 19     98    69
## 20     99    28
## 21    100    19
```

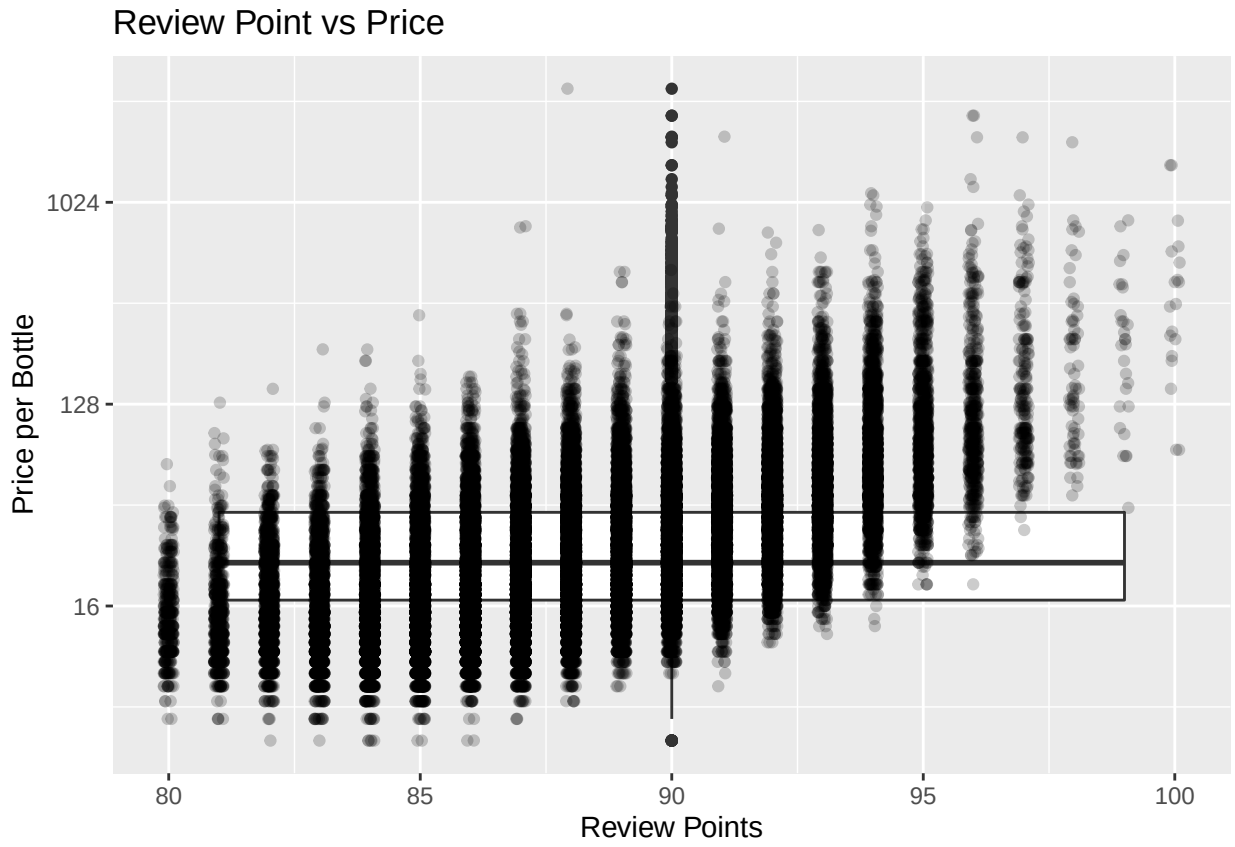
```r
#*********************************************************************************
# Plot a bar chart of the number of bottles vs review points
#*********************************************************************************

p<- ggplot(wine_review, aes(factor(points)))
p + geom_bar(aes(fill = factor(points))) +
theme(legend.position = "none") +
xlab("Poinits") +
ylab("Number of Bottle") +
ggtitle("Number of Wines per Points")
```




```
#*********************************************************************************
# Determine the price of the top ten common bottle counts
#*********************************************************************************
wine_review%>%
group_by(price) %>%
summarize(n = n())%>%
arrange(desc(n), price)%>%
print(n=10)
```

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

## # A tibble: 390 x 2
##    price     n
##    <dbl> <int>
##  1    20  6938
##  2    15  6066
##  3    25  5796
##  4    30  4946
##  5    18  4881
##  6    12  3932
##  7    40  3871
##  8    35  3801
##  9    13  3548
## 10    16  3545
## # ... with 380 more rows
```

```
#*********************************************************************************
# Plot a boxplot of points vs price distribution of wines
#*********************************************************************************

wine_review%>%
ggplot(aes(points, price, group = 1)) +
geom_boxplot()+
geom_jitter(width = 0.1, alpha = 0.2) +
scale_y_continuous(trans = "log2") +
xlab("Review Points") +
ylab("Price per Bottle") +
ggtitle("Review Point vs Price")
```




```
# Define rating value of each rows that corresponds to the point value.
```

```
c5<- wine_review%>%filter(points >=95 & points <=100)%>%
mutate(rating = 5)
c4<- wine_review%>%filter(points >=90 & points <=94) %>%
mutate(rating = 4)
c3<- wine_review%>%filter(points >=85 & points <=89) %>%
mutate(rating = 3)
c2<- wine_review%>%filter(points >=80 & points <=84) %>%
mutate(rating = 2)
c1<- wine_review%>%filter(points >=75 & points <=79) %>%
mutate(rating = 1)
```

```
c0<- wine_review%>%filter(points >=50 & points <=74) %>%
mutate(rating = 0)
```

```
#*********************************************************************************
# Combine all rated rows into wine_review dataframe.
#*********************************************************************************

t1<-rbind(c5,c4)
t2<-rbind(t1,c3)
t3<-rbind(t2,c2)

wine_review<-t3
```

```
#*********************************************************************************
# Define country column numerically -
# Create column countryId, and assign corresponding numerical value of
# each country.
#*********************************************************************************

a1001<- wine_review%>%filter(country == 'Argentina')%>%mutate(countryId = 1001)
a1002<- wine_review%>%filter(country == 'Armenia')%>%mutate(countryId = 1002)
a1003<- wine_review%>%filter(country == 'Australia')%>%mutate(countryId = 1003)
a1004<- wine_review%>%filter(country == 'Austria')%>%mutate(countryId = 1004)
a1005<- wine_review%>%filter(country == 'Bosnia and Herzegovina')%>%mutate(countryId = 1005)
a1006<- wine_review%>%filter(country == 'Brazil')%>%mutate(countryId = 1006)
a1007<- wine_review%>%filter(country == 'Bulgaria')%>%mutate(countryId = 1007)
a1008<- wine_review%>%filter(country == 'Canada')%>%mutate(countryId = 1008)
a1009<- wine_review%>%filter(country == 'Chile')%>%mutate(countryId = 1009)
a1010<- wine_review%>%filter(country == 'China')%>%mutate(countryId = 1010)
a1011<- wine_review%>%filter(country == 'Croatia')%>%mutate(countryId = 1011)
a1012<- wine_review%>%filter(country == 'Cyprus')%>%mutate(countryId = 1012)
a1013<- wine_review%>%filter(country == 'Czech Republic')%>%mutate(countryId = 1013)
a1014<- wine_review%>%filter(country == 'England')%>%mutate(countryId = 1014)
a1015<- wine_review%>%filter(country == 'France')%>%mutate(countryId = 1015)
a1016<- wine_review%>%filter(country == 'Georgia')%>%mutate(countryId = 1016)
a1017<- wine_review%>%filter(country == 'Germany')%>%mutate(countryId = 1017)
a1018<- wine_review%>%filter(country == 'Greece')%>%mutate(countryId = 1018)
a1019<- wine_review%>%filter(country == 'Hungary')%>%mutate(countryId = 1019)
a1020<- wine_review%>%filter(country == 'India')%>%mutate(countryId = 1020)
a1021<- wine_review%>%filter(country == 'Israel')%>%mutate(countryId = 1021)
a1022<- wine_review%>%filter(country == 'Italy')%>%mutate(countryId = 1022)
a1023<- wine_review%>%filter(country == 'Lebanon')%>%mutate(countryId = 1023)
a1024<- wine_review%>%filter(country == 'Luxembourg')%>%mutate(countryId = 1024)
a1025<- wine_review%>%filter(country == 'Macedonia')%>%mutate(countryId = 1025)
a1026<- wine_review%>%filter(country == 'Mexico')%>%mutate(countryId = 1026)
a1027<- wine_review%>%filter(country == 'Moldova')%>%mutate(countryId = 1027)
a1028<- wine_review%>%filter(country == 'Morocco')%>%mutate(countryId = 1028)
a1029<- wine_review%>%filter(country == 'New Zealand')%>%mutate(countryId = 1029)
a1030<- wine_review%>%filter(country == 'Peru')%>%mutate(countryId = 1030)
a1031<- wine_review%>%filter(country == 'Portugal')%>%mutate(countryId = 1031)
a1032<- wine_review%>%filter(country == 'Romania')%>%mutate(countryId = 1032)
a1033<- wine_review%>%filter(country == 'Serbia')%>%mutate(countryId = 1033)
a1034<- wine_review%>%filter(country == 'Slovakia')%>%mutate(countryId = 1034)
```

```
a1035<- wine_review%>%filter(country == 'Slovenia')%>%mutate(countryId = 1035)
a1036<- wine_review%>%filter(country == 'South Africa')%>%mutate(countryId = 1036)
a1037<- wine_review%>%filter(country == 'Spain')%>%mutate(countryId = 1037)
a1038<- wine_review%>%filter(country == 'Switzerland')%>%mutate(countryId = 1038)
a1039<- wine_review%>%filter(country == 'Turkey')%>%mutate(countryId = 1039)
a1040<- wine_review%>%filter(country == 'Ukraine')%>%mutate(countryId = 1040)
a1041<- wine_review%>%filter(country == 'Uruguay')%>%mutate(countryId = 1041)
a1042<- wine_review%>%filter(country == 'US')%>%mutate(countryId = 1042)
```

```
#*******************************************************************************
# Combine all temporary tables created to assign country id into a single dataframe,
# and update wine_review dataframe.
#*******************************************************************************
country_id <- rbind (a1001,a1002,a1003,a1004,a1005,a1006,a1007,a1008,a1009,a1010,
                     a1011,a1012,a1013,a1014,a1015,a1016,a1017,a1018,a1019,a1020,
                     a1021,a1022,a1023,a1024,a1025,a1026,a1027,a1028,a1029,a1030,
                     a1031,a1032,a1033,a1034,a1035,a1036,a1037,a1038,a1039,a1040,
                     a1041,a1042)

wine_review<-country_id
```

```
#*******************************************************************************
# Examine/confirm updated dataframe observations and variables; and list the
# columns name.
#*******************************************************************************
dim(wine_review)
```

```
## [1] 120916      6
```

```
names(wine_review)
```

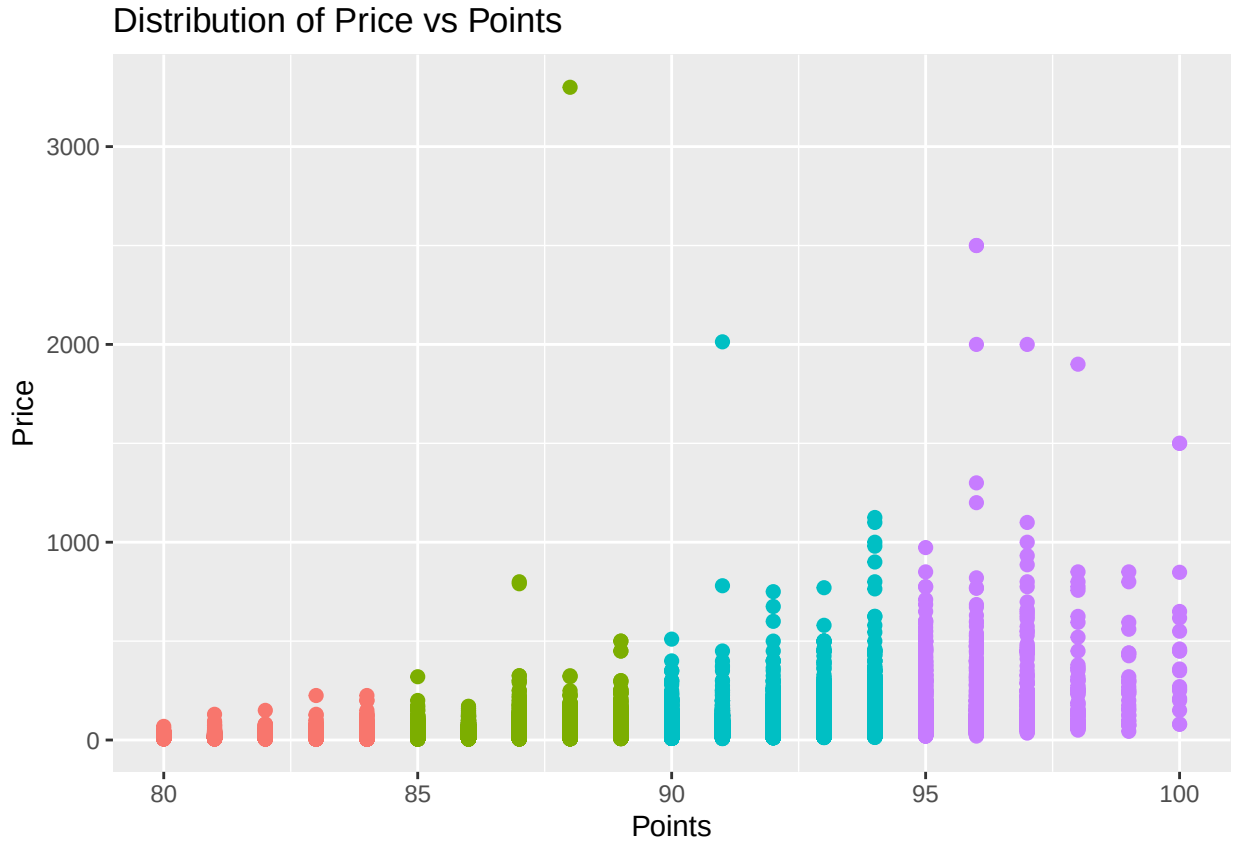
```
## [1] "wineId"    "country"   "points"    "price"     "rating"    "countryId"
```

```
#*******************************************************************************
#Determine the count of different wines, maximum price, minimum price, and
# average price in each rating group
#*******************************************************************************

wine_review %>%
group_by("Rating" = rating) %>%
summarize("Count" = n(),
                "Max. Price" = max(price),
                "Min, Price" = min(price),
                "Avg. Price" = round(mean(price)))%>%
knitr::kable()
```

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

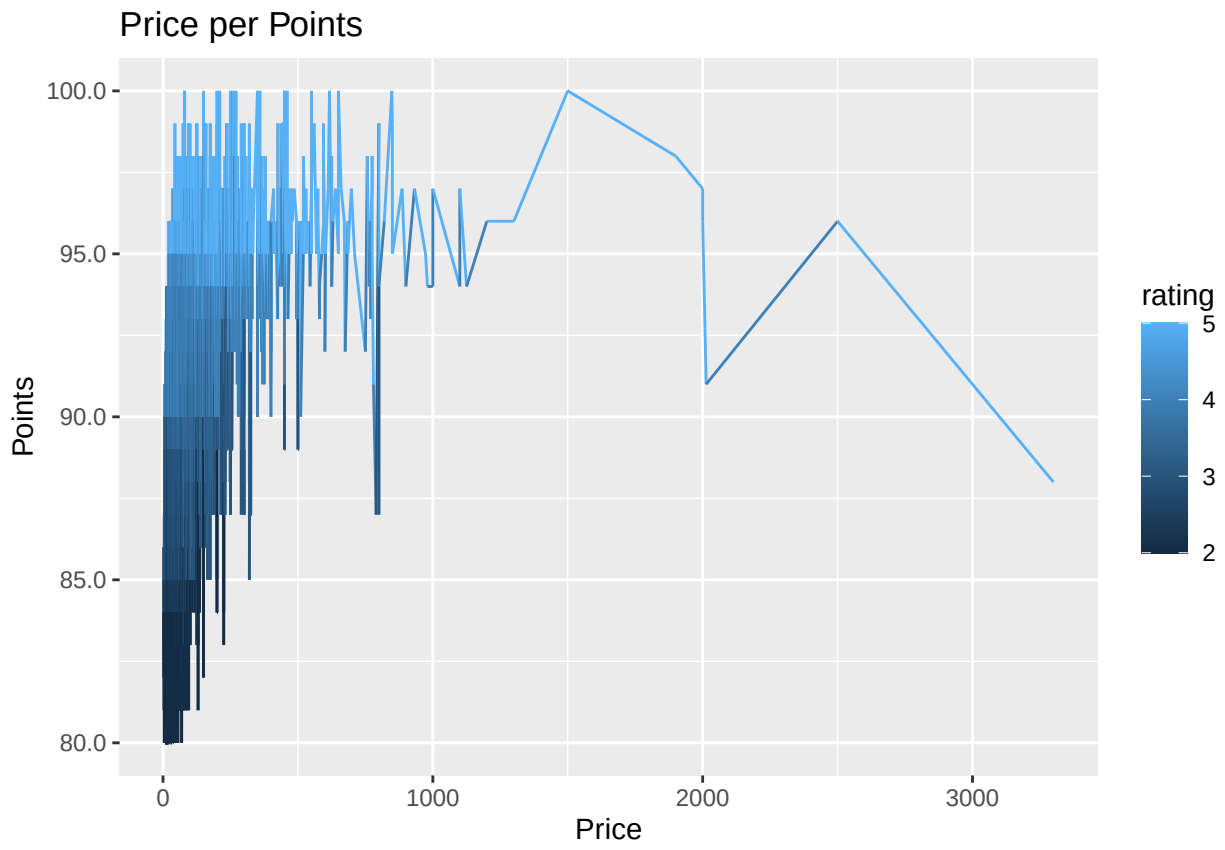
Rating	Count	Max. Price	Min, Price	Avg. Price
2	11830	225	4	19
3	63713	3300	4	26
4	43162	2013	7	49
5	2211	2500	20	139

```
#*******************************************************************************
# Plot - Distribution of Points vs Price
#*******************************************************************************

ggplot(wine_review, aes(x = points, y = price)) +
geom_point(aes(colour = factor(rating)), size = 2) +
theme(legend.position = "none") +
xlab("Points") +
ylab("Price") +
ggtitle("Distribution of Price vs Points")
```




```
#*******************************************************************************
# Plot - a line graph related to distribution of Price vs Points line graph
#*******************************************************************************
wine_review%>%
filter(rating%in%c("5", "4", "3", "2")) %>%
ggplot(aes(price, points, col = rating)) +
geom_line() +
```

```
scale_y_continuous(name = "Points", labels = scales::comma) +
xlab("Price") +
ggtitle("Price per Points")
```




```
#################################################################################
# Since linear regression is sensitive to outliers; we will identify price anomaly
# for group 3 and 4 and remove the extreme data.
#################################################################################
wine_review%>%
filter(rating == 3) %>%
arrange(desc(price)) %>%
print(n=1)
```

```
## # A tibble: 63,713 x 6
##   wineId country points price rating countryId
##    <dbl> <chr>    <dbl> <dbl>  <dbl>     <dbl>
## 1  80290 France      88  3300      3      1015
## # ... with 63,712 more rows
```

```
wine_review%>%
filter(rating == 4) %>%
arrange(desc(price)) %>%
print(n=1)
```

```
## # A tibble: 43,162 x 6
##   wineId country points price rating countryId
##    <dbl> <chr>    <dbl> <dbl>  <dbl>     <dbl>
## 1 120391 US          91  2013      4      1042
## # ... with 43,161 more rows
```

```
#*********************************************************************************
# Drop anomaly data from rating 3 and 4
#*********************************************************************************
wine_review<-subset(wine_review, wineId!=80290 & wineId!=120391)
```

```
#*********************************************************************************
# Plot graph to observe irregular data are removed
#*********************************************************************************
p <- ggplot(wine_review, aes(rating, price))
p + geom_point(aes(colour = factor(rating)), size = 2)+
theme(legend.position = "none") +
xlab("Rating") +
ylab("Price") +
ggtitle("Distribution of Price per Rating")
```


```
#*********************************************************************************
# Set seed to 1.
# Extract 10% of dataset for validation purpose, and assign the remaining 90%
# to wine_df dataframe.
#*********************************************************************************
```

```r
set.seed(1, sample.kind = "Rounding")
```

```r
y = wine_review$rating
test_index<- createDataPartition(y, times = 1, p = 0.1, list = FALSE)
wine_df<- wine_review[-test_index,]
temp<- wine_review[test_index,]
```

```r
#################################################################################
# Create dataframe validation, and make sure country and price
# are also in wine_df dataset
#################################################################################
validation<- temp %>%
  semi_join(wine_df, by = "country") %>%
  semi_join(wine_df, by = "price")
```

```r
# Determine row count of validation dataset
nrow(validation)
```

```
## [1] 12082
```

```r
#################################################################################
# Keep data from temp table for which there are no data in
# validation dataset; and determine row count of removed rows
#################################################################################

removed <- anti_join(temp, validation)
```

```
## Joining, by = c("wineId", "country", "points", "price", "rating", "countryId")
```

```r
nrow(removed)
```

```
## [1] 12
```

```r
#################################################################################
# add rows removed from validation table back into wine_df table
#################################################################################
wine_df <- rbind(wine_df, removed)

nrow(wine_df)
```

```
## [1] 108832
```

```r
#################################################################################
# Set seed value to 1, create training and testing datasets from wine_df dataset.
# 90% for training, and 10% for testing.
#################################################################################
```

```r
set.seed(1, sample.kind="Rounding")
```

```r
train_index <- createDataPartition(wine_df$rating, times = 1, p = 0.1, list = FALSE)
train_set <- wine_df[-train_index,]
temp_set <- wine_df[train_index,]
```

```r
#*******************************************************************************
# Make sure country and price in test_set are in train_set
#*******************************************************************************
test_set <- temp_set %>%
  semi_join(train_set, by = "country") %>%
  semi_join(train_set, by = "price")
```

```r
#*******************************************************************************
# Create dataframe removed to populate with data not available in test_set.
# Determine the count of rows extracted in removed table
#*******************************************************************************
removed <- anti_join(temp_set, test_set)
```

```
## Joining, by = c("wineId", "country", "points", "price", "rating", "countryId")
```

```r
nrow(removed)
```

```
## [1] 10
```

```r
#*******************************************************************************
# Combine rows from removed dataframe, back into train_set; and determine the number
# of rows in train_set dataset
#*******************************************************************************
train_set <- rbind(train_set, removed)
nrow(train_set)
```

```
## [1] 97958
```

```r
#*******************************************************************************
# Determine total number of rows in train and test set:
#*******************************************************************************
nrow(train_set) + nrow(test_set)
```

```
## [1] 108832
```

```r
#*******************************************************************************
#                           Develop Models for Prediction
#*******************************************************************************
#*******************************************************************************
# 1st model: Simple Average Model - Baseline Model
#*******************************************************************************

# mu - average rating of all bottle of wines.

mu <- mean(train_set$rating)

model_1_rmse <- RMSE(test_set$rating, mu)
model_1_rmse
```

```
## [1] 0.6638091
```

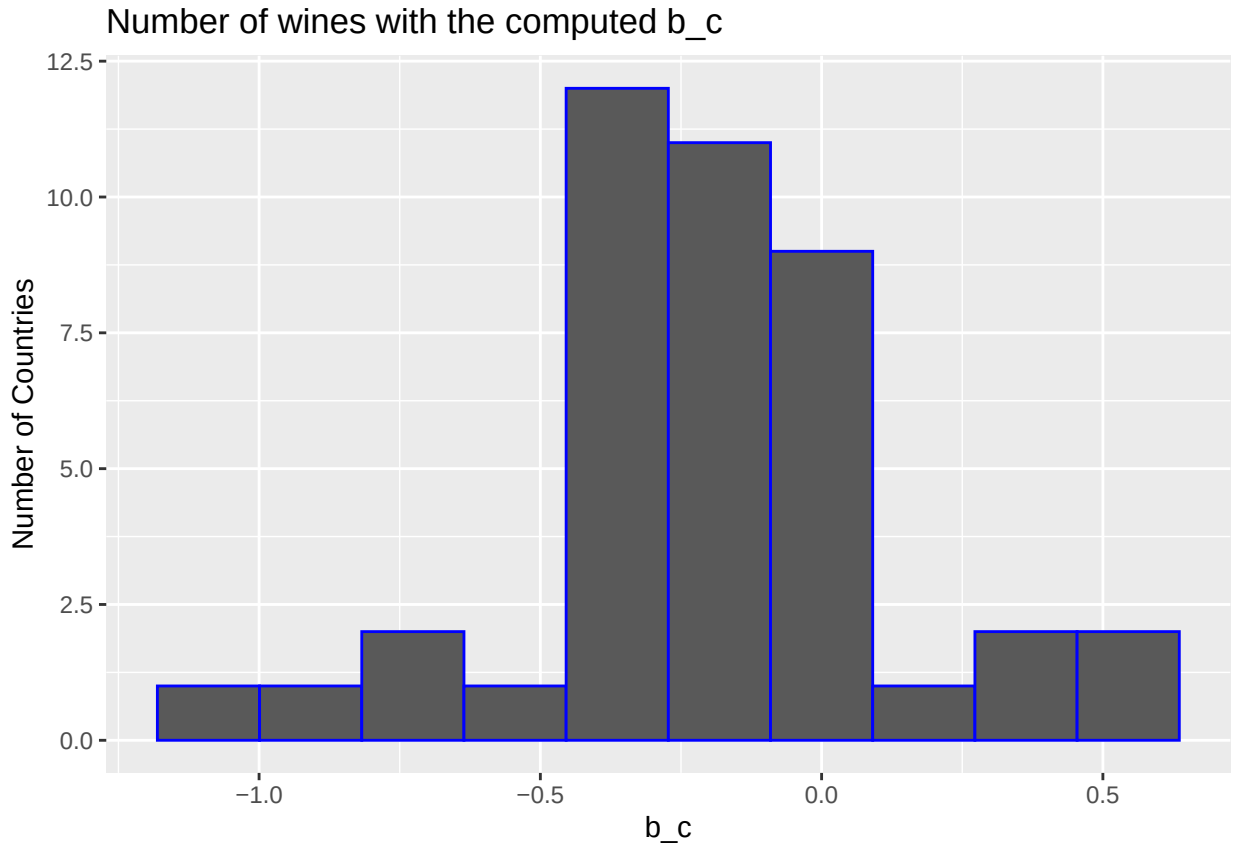
```
#*****************************************************************************
# create dataframe(table), rmse_results to store the results of RMSE values.
#*****************************************************************************

rmse_results <- data_frame(Model = "Simple Average", RMSE = model_1_rmse)

rmse_results%>%knitr::kable()
```

Model	RMSE
Simple Average	0.6638091

```
#*****************************************************************************
# 2nd model: Country Effect Model
# Improve model by adding country effect parameter, b_c, that represents the
# average rating for country:
#*****************************************************************************

coun_avgs <- train_set %>%
  group_by(countryId) %>%
  summarize(b_c = mean(rating - mu))
```

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

```
coun_avgs %>%
  qplot(b_c, geom ="histogram", bins = 10, data = ., color = I("blue"),
        ylab = "Number of Countries",
        main = "Number of wines with the computed b_c")
```




```
predicted_ratings <- mu + test_set %>%
  left_join(coun_avgs, by='countryId') %>%
  .$b_c

model_2_rmse <- RMSE(predicted_ratings, test_set$rating)

rmse_results <- bind_rows(rmse_results,
                          data_frame(Model="Country Effect Model", RMSE = model_2_rmse ))

# Update the results table
rmse_results %>%knitr::kable()
```

Model	RMSE
Simple Average	0.6638091
Country Effect Model	0.6473962

```
#********************************************************************************
# 3rd model: Country and Price Effect Model
# Improve model by adding price effect (b_p):
#
#********************************************************************************

# Determine for each country average wine price, average rating, and count wines
```

```r
# produced in the aggregated dataset.

wine_corr<- wine_review%>%
group_by(countryId) %>%
summarize(Avg_Price = round(mean(price)),
                Mean_rating = round(mean(rating)),
                Count = n())%>%
arrange(Avg_Price)%>%
print(n=43)
```

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

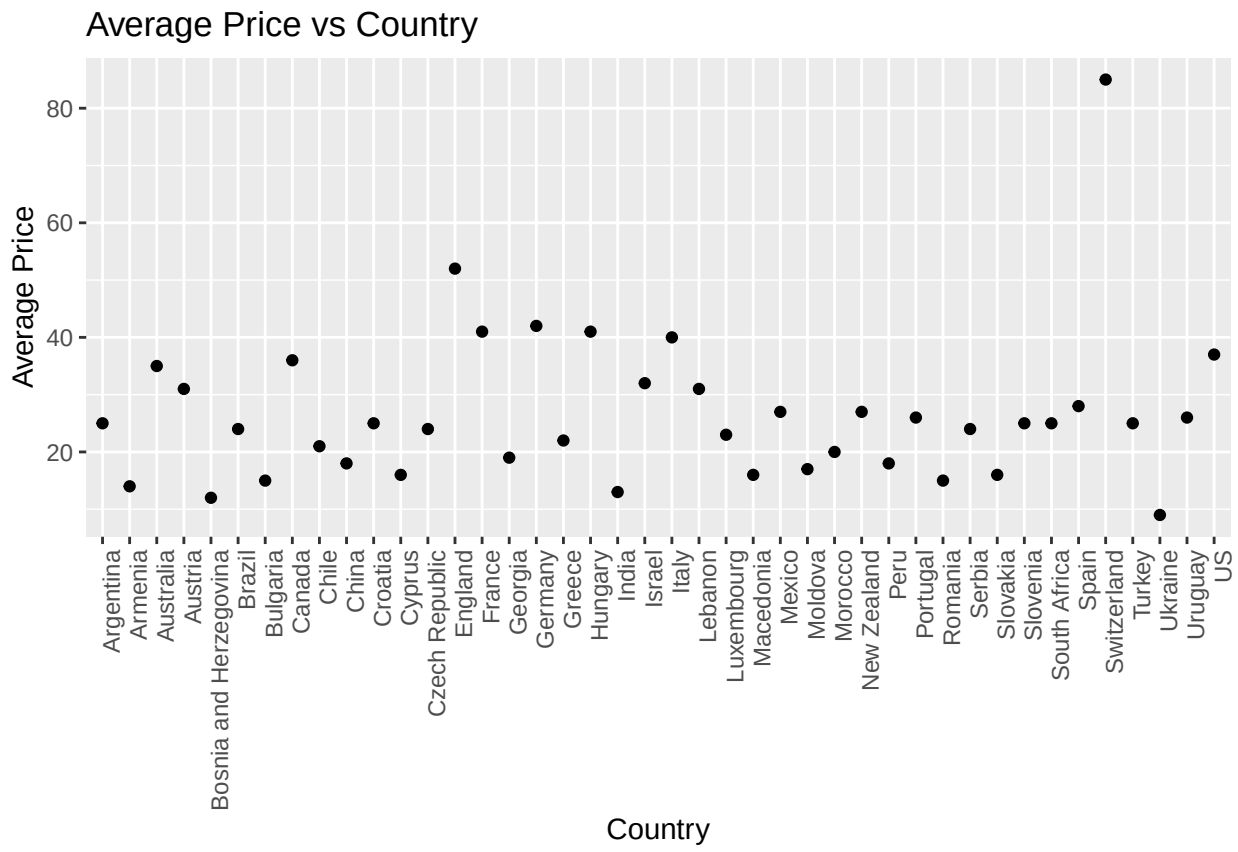
```
## # A tibble: 42 x 4
##    countryId Avg_Price Mean_rating Count
##        <dbl>     <dbl>       <dbl> <int>
##  1      1040         9           2    14
##  2      1005        12           3     2
##  3      1020        13           4     9
##  4      1002        14           3     2
##  5      1007        15           3   141
##  6      1032        15           3   120
##  7      1012        16           3    11
##  8      1025        16           3    12
##  9      1034        16           3     1
## 10      1027        17           3    59
## 11      1010        18           3     1
## 12      1030        18           2    16
## 13      1016        19           3    84
## 14      1028        20           3    28
## 15      1009        21           3  4416
## 16      1018        22           3   461
## 17      1024        23           3     6
## 18      1006        24           3    47
## 19      1013        24           3    12
## 20      1033        24           3    12
## 21      1001        25           3  3756
## 22      1011        25           3    71
## 23      1035        25           3    80
## 24      1036        25           3  1293
## 25      1039        25           3    90
## 26      1031        26           3  4875
## 27      1041        26           3   109
## 28      1026        27           3    70
## 29      1029        27           3  1378
## 30      1037        28           3  6573
## 31      1004        31           4  2799
## 32      1023        31           3    35
## 33      1021        32           3   489
## 34      1003        35           3  2294
## 35      1008        36           4   254
## 36      1042        37           3 54264
## 37      1022        40           3 16914
## 38      1015        41           3 17775
```

```
## 39       1019         41            3   145
## 40       1017         42            4  2120
## 41       1014         52            4    69
## 42       1038         85            3     7
```

```
#************************************************************
# Plot average wine price vs country
#************************************************************
```

```
wine_review%>%
group_by(country) %>%
summarize(price = round(mean(price))) %>%
ggplot(aes(country, price)) +
geom_point() +
ggtitle("Average Price vs Country") +
xlab("Country") +
ylab("Average Price") +
theme(axis.text.x = element_text(angle = 90, hjust = 1))
```

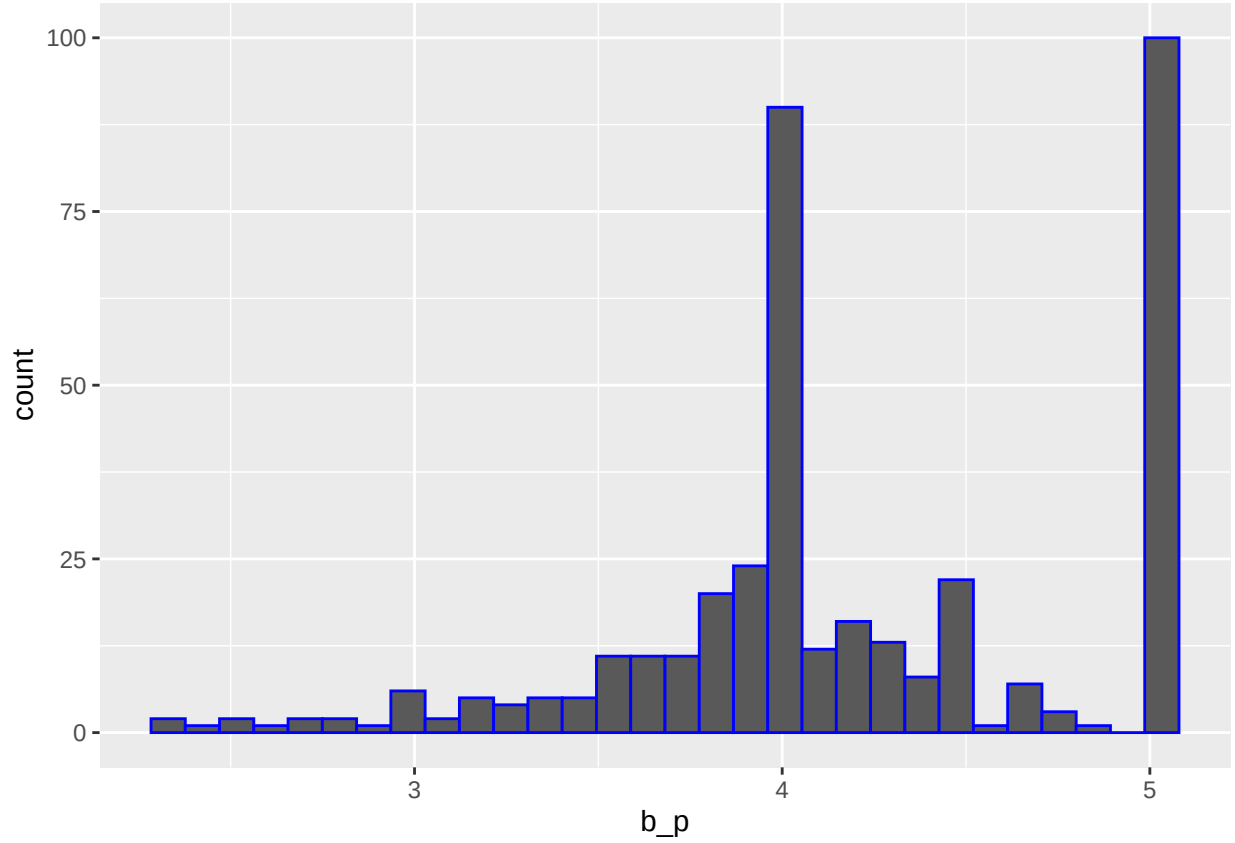
```
## `summarise()` ungrouping output (override with `.groups` argument)
```




```
# Different wines are rated differently even though the prices are the same.
# For further insight to the dataset, we will compute the average price for
# each country that have produced wines.
```

```
# Plot a histogram graph to see trend.

train_set %>%
  group_by(price) %>%
  summarize(b_p = mean(rating)) %>%
  ggplot(aes(b_p)) +
  geom_histogram(bins = 30, color = "blue")
```

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




```
# The generated histogram above shows that the rating data are not normally
# distributed. Let us see if country + price model will provide lower RMSE value.
```

```
price_avgs <- train_set %>%
  left_join(coun_avgs, by='countryId') %>%
  group_by(price) %>%
  summarize(b_p = mean(rating - mu - b_c))
```

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

```
predicted_ratings <- test_set %>%
  left_join(coun_avgs, by='countryId') %>%
  left_join(price_avgs, by='price') %>%
```

```
  mutate(pred = mu + b_c + b_p) %>%
  .$pred

model_3_rmse <- RMSE(predicted_ratings, test_set$rating)

rmse_results <- bind_rows(rmse_results,
      data_frame(Model="Country + Price Effect Model", RMSE = model_3_rmse ))
```

```
# Update the results table
rmse_results %>%
knitr::kable()
```

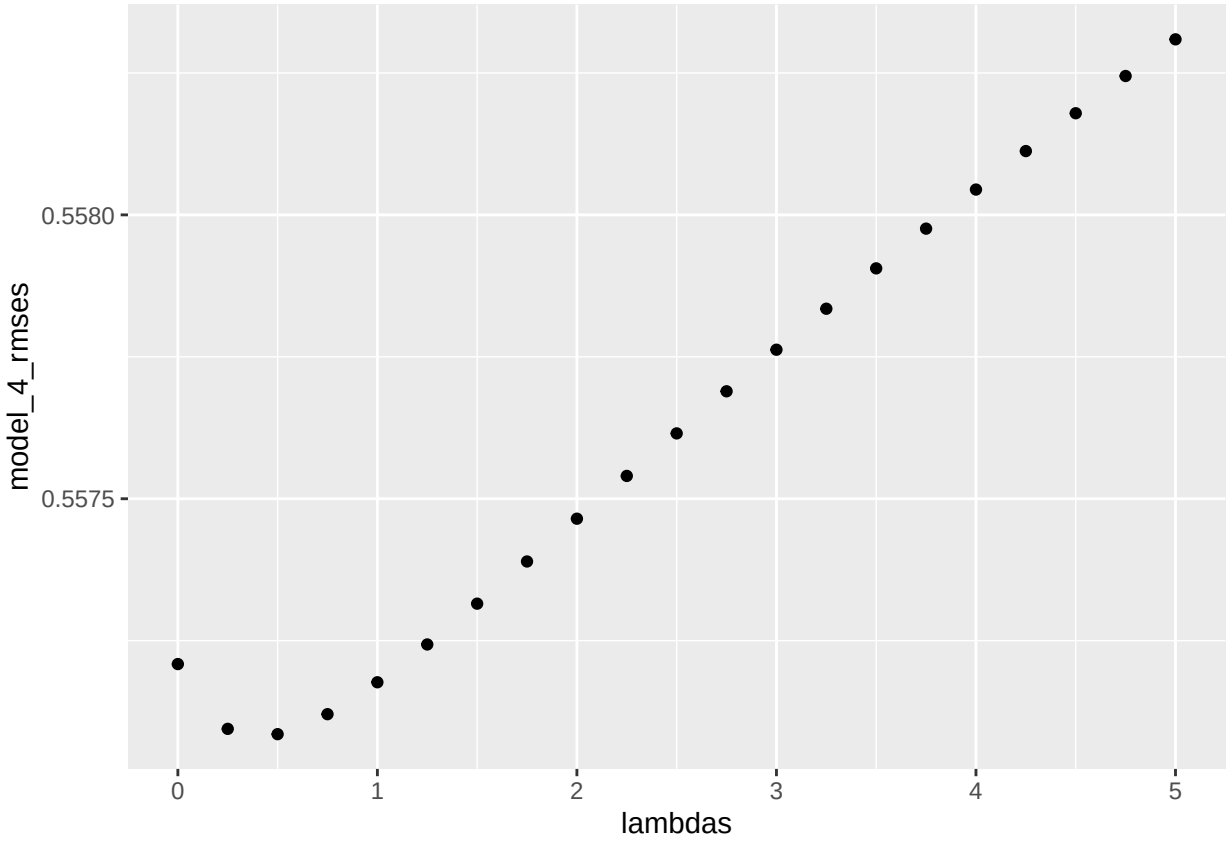
Model	RMSE
Simple Average	0.6638091
Country Effect Model	0.6473962
Country + Price Effect Model	0.5572087

```
# we have achieved to lower the RMSE value.
```

```
#*******************************************************************************
# 4th Model: Country + Price Effect Model
# By tuning the model, let us determine lambda value that will minimize
# RMSE value for country + price effect.
#*******************************************************************************
```

```
lambdas <- seq(0, 5, 0.25)
model_4_rmses <- sapply(lambdas, function(lambdas){
  mu <- mean(train_set$rating)
  b_c <- train_set %>%
    group_by(countryId) %>%
    summarize(b_c = sum(rating - mu)/(n()+lambdas))
  b_p <- train_set %>%
    left_join(b_c, by="countryId") %>%
    group_by(price) %>%
    summarize(b_p = sum(rating - b_c - mu)/(n()+lambdas))
  predicted_ratings <- test_set %>%
    left_join(b_c, by = "countryId") %>%
    left_join(b_p, by = "price") %>%
    mutate(pred = mu + b_c + b_p) %>%
    .$pred
  return(RMSE(predicted_ratings, test_set$rating))
})
```

```
qplot(lambdas, model_4_rmses)
```




```
# The generated plot above shows the optimal lambda value using regularized
# country + price effects.
```

```
# Using the minimum lambda value obtained determine RMSE value.

lambda <- lambdas[which.min(model_4_rmses)]
lambda
```

```
## [1] 0.5
```

```
rmse_results <- bind_rows(rmse_results,
                          data_frame(Model="Regularized Country + Price Effect Model",
                                     RMSE = min(model_4_rmses)))
```

```
# Update the results table
rmse_results %>%  knitr::kable()
```

Model	RMSE
Simple Average	0.6638091
Country Effect Model	0.6473962
Country + Price Effect Model	0.5572087
Regularized Country + Price Effect Model	0.5570853

```
#*********************************************************************************
#
# Evaluating predictive model using wine_df (training dataset)
# and validation (validation dataset)
#
#*********************************************************************************
#
# A model is best validated with independent large training datasets. Therefore,
# we will use wine_df dataset (train_set + test_set) to train the model, and use
# the validation dataset to evaluate the prediction, and obtain RSME value.
#
#*********************************************************************************
#
# 5th Model: Evaluating country + price Effect Model
#
# Since the country + price Effect Model produced a lower RSME value than the country
# Effect Model, we will use derived minimum lambda value to compute the
# regularized effect and achieve lower RSME result.
#
#*********************************************************************************
```

```
mu_vali <- mean(wine_df$rating)

country_avgs_vali <- wine_df %>%
  group_by(countryId) %>%
  summarize(b_c = sum(rating - mu_vali)/(n()+lambda))
```

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

```
price_avgs_vali <- wine_df %>%
  left_join(country_avgs_vali, by='countryId') %>%
  group_by(price) %>%
  summarize(b_p = sum(rating - b_c - mu_vali)/(n()+lambda))
```

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

```
predicted_vali <- validation %>%
  left_join(country_avgs_vali, by='countryId') %>%
  left_join(price_avgs_vali, by='price') %>%
  mutate(pred = mu_vali + b_c + b_p) %>%
  .$pred

validate_model <- RMSE(predicted_vali, validation$rating)

rmse_results <- bind_rows(rmse_results,
            data_frame(Model="Validating Regularized Country + Price Effect Model",
                                     RMSE = validate_model))
```

```
# Update the results table
rmse_results %>%
knitr::kable()
```

Model	RMSE
Simple Average	0.6638091
Country Effect Model	0.6473962
Country + Price Effect Model	0.5572087
Regularized Country + Price Effect Model	0.5570853
Validating Regularized Country + Price Effect Model	0.5595011

```
#*****************************************************************************
#
#                              Matrix Factorization
#
#*****************************************************************************
```

Recosystem is an R wrapper of the LIBMF library developed by Yu-Chin Juan, Yong Zhuang, Wei-Sheng Chin and Chih-Jen Lin (http://www.csie.ntu.edu.tw/~cjlin/libmf/), an open source library for recommender system using matrix factorization.

LIBMF is a parallelized library, that users can take advantage of multicore CPUs to speed up the computation. It also utilizes some advanced CPU features to further improve the performance. (Lin et al. 2014)

```
# Usage of recosystem
```

The usage of recosystem is quite simple, mainly consisting of the following steps:

1. Create a model object (a Reference Class object in R) by calling Reco().
2. (Optionally) call the $tune() method to select best tuning parameters along a set of candidate values.
3. Train the model by calling the $train() method. A number of parameters can be set inside the function, possibly coming from the result of $tune().
4. (Optionally) output the model, i.e. write the factorized P and Q matrices info files.
5. Use the $predict() method to compute predictions and write results into a file.

```
#*****************************************************************************
#
# Set seed to 1,
# and assign training dataset as train_mf, and testing dataset as test_mf
#
#*****************************************************************************
```

```
set.seed(1, sample.kind = "Rounding")
```

```
# Convert the train and test sets into recosystem input format
train_mf <-with(train_set, data_memory(user_index = wineId,
                item_index = price, rating = rating))
test_mf <-with(test_set, data_memory(user_index = wineId,
                item_index = price, rating = rating))


# Create the model object
r <-  recosystem::Reco()
```

```
# Tune model, Select the best tuning parameters
opts <- r$tune(train_mf, opts = list(dim = c(10, 20, 30),
                                     lrate = c(0.1, 0.2),
                                     costp_l2 = c(0.01, 0.1),
                                     costq_l2 = c(0.01, 0.1),
                                     nthread  = 4, niter = 10))

 # Train the algorithm
r$train(train_mf, opts = c(opts$min, nthread = 4, niter = 20))
```

```
## iter      tr_rmse          obj
##    0       1.1054  2.2535e+005
##    1       2.0393  4.8640e+005
##    2       1.0080  1.7829e+005
##    3       0.7919  1.3938e+005
##    4       0.6582  1.1918e+005
##    5       0.5637  1.0706e+005
##    6       0.4885  9.8049e+004
##    7       0.4310  9.1642e+004
##    8       0.3848  8.6900e+004
##    9       0.3466  8.3079e+004
##   10       0.3119  7.9873e+004
##   11       0.2868  7.7442e+004
##   12       0.2611  7.5276e+004
##   13       0.2427  7.3551e+004
##   14       0.2247  7.2103e+004
##   15       0.2089  7.0801e+004
##   16       0.1955  6.9739e+004
##   17       0.1850  6.9020e+004
##   18       0.1744  6.8142e+004
##   19       0.1674  6.7600e+004
```

```
# Calculate the predicted values
pr_reco <-  r$predict(test_mf, out_memory())

rmse_results <- bind_rows(rmse_results,
                  data_frame(Model = "Matrix Factorization",
                          RMSE = RMSE(test_set$rating, pr_reco)))
```

```
# Update the results table
rmse_results %>%
knitr::kable()
```

Model	RMSE
Simple Average	0.6638091
Country Effect Model	0.6473962
Country + Price Effect Model	0.5572087
Regularized Country + Price Effect Model	0.5570853
Validating Regularized Country + Price Effect Model	0.5595011
Matrix Factorization	0.6638091

```
#*****************************************************************************
#
#                              Matrix Factorization - Validation
#
#*****************************************************************************
```

```
set.seed(1, sample.kind = "Rounding")
```

```
# Convert 'wine_df' and 'validation' sets to recosystem input format
wine_reco <-  with(wine_df,
              data_memory(user_index = wineId, item_index = price,
                          rating = rating))
validation_reco <-  with(validation, data_memory(user_index = wineId,
                         item_index = price, rating = rating))


# Create the model object
r_vali <-  recosystem::Reco()

# Tune the parameters
opts_vali <-r_vali$tune(wine_reco, opts = list(dim = c(10, 20, 30),
                                    lrate = c(0.1, 0.2),
                                    costp_l2 = c(0.01, 0.1),
                                    costq_l2 = c(0.01, 0.1),
                                    nthread  = 4, niter = 10))

# Train the model
r_vali$train(wine_reco, opts = c(opts_vali$min, nthread = 4, niter = 20))
```

```
## iter      tr_rmse          obj
##    0       1.0724  2.4154e+005
##    1       2.0567  5.4816e+005
##    2       1.0122  1.9860e+005
##    3       0.7847  1.5346e+005
##    4       0.6513  1.3149e+005
##    5       0.5541  1.1741e+005
##    6       0.4769  1.0742e+005
##    7       0.4215  1.0089e+005
##    8       0.3780  9.5712e+004
##    9       0.3416  9.1707e+004
##   10       0.3106  8.8470e+004
##   11       0.2835  8.5655e+004
##   12       0.2617  8.3417e+004
##   13       0.2421  8.1551e+004
##   14       0.2254  8.0067e+004
##   15       0.2103  7.8605e+004
##   16       0.1980  7.7567e+004
##   17       0.1856  7.6429e+004
##   18       0.1778  7.5755e+004
##   19       0.1703  7.5049e+004
```

```
# Calculate the prediction
pr_reco_vali<-r_vali$predict(validation_reco, out_memory())

# Update the result table
rmse_results <- bind_rows(rmse_results,
                data_frame(Model = "Validating Matrix Factorization",
                      RMSE = RMSE(validation$rating, pr_reco_vali)))
```

```
#*****************************************************************************
# Results
#*****************************************************************************
```

```
rmse_results %>%knitr::kable()
```

Model	RMSE
Simple Average	0.6638091
Country Effect Model	0.6473962
Country + Price Effect Model	0.5572087
Regularized Country + Price Effect Model	0.5570853
Validating Regularized Country + Price Effect Model	0.5595011
Matrix Factorization	0.6638091
Validating Matrix Factorization	0.6679870

```
#*****************************************************************************
# Conclusion
#*****************************************************************************
```

Based on RMSE values obtained, the regularized linear regression model has achieved the lowest value of 0.5570853 when the country of wine produced and price effect was used. Using the validation sample, the RMSE value achieved was 0.5595011.

We can infer from this report, the regularized linear regression is a better recommendation system.

This project has demonstrated the lesson learned from the HarvardX's Data Science education. For simplicity and computation time constraint reasons, the analysis in this project utilized country and price effect. Other effects and better models can be utilized to obtain better RMSE value. By combining different machine learning models and using ensembles, we can further improve the predictions result.

```
#*****************************************************************************
# Reference
#*****************************************************************************
```


1 - https://www.edx.org/professional-certificate/harvardx-data-science

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

3 - http://cran.us.r-project.org

4 - https://www.kaggle.com/zynicide/wine-reviews

5 - https://www.marketviewliquor.com/blog/2020/01/how-does-the-wine-rating-system-work/

6 - https://en.wikipedia.org/wiki/Wine

7 - https://www.rdocumentation.org/packages/recosystem/versions/0.3

8 - http://www.csie.ntu.edu.tw/~cjlin/libmf/

```
#*****************************************************************************
# Appendix -
#*****************************************************************************
```

Note :

Zip file, wine_review_data.csv.zip, contains the dataset, and must reside in the working directory before executing Rmd or R script.

Different versions of R or Rstudio could provide slightly different RMSE value.

The version of software used for generating the RMSE values in this report are:

Rstudio Version 1.2a.5033

R version 3.6.2 (2019-12-12)

```
print("Operating System:")
```

```
## [1] "Operating System:"
```

```
version
```

```
##                _
## platform       x86_64-w64-mingw32
## arch           x86_64
## os             mingw32
## system         x86_64, mingw32
## status
## major          3
## minor          6.2
## year           2019
## month          12
## day            12
## svn rev        77560
## language       R
## version.string R version 3.6.2 (2019-12-12)
## nickname       Dark and Stormy Night
```