```
# PREDICT USER'S STAR RATING BASED ON DATA ON THE USER PROFILE AND BUSINESS ATTRIBUTES

#Load packages
library(dplyr)
library(tidyverse)
library(readr)
library(jsonlite)
library(caret)
library(stringr)
library(glmnet)
library(lubridate)

#Clear
cat("\014")
rm(list=ls())

#Set Directory as appropriate
setwd("C:/Users/angel/OneDrive - University of Warwick/Year 3/EC349 Data Science/R
projects/EC349-assignment/Yelp-datasets")

#Load .json Data
business_data <- stream_in(file("yelp_academic_dataset_business.json"))
checkin_data  <- stream_in(file("yelp_academic_dataset_checkin.json"))
tip_data  <- stream_in(file("yelp_academic_dataset_tip.json"))

#Load small data
review_data  <- load(file='yelp_review_small.Rda')
user_data  <- load(file='yelp_user_small.Rda')

#--------------------------------------------------------------------------------#

### 1. BUSINESS DATA
#Keep relevant business info
business_data <- business_data %>%
  select(business_id, name, city, state, postal_code, stars, review_count, categories,
hours)

#Extract hours dataframe
hours <- flatten(business_data$hours)

#Is the business open on weekends (both days)?
hours <- hours %>%
  mutate(wkend_open = ifelse(!is.na(Saturday) & Saturday != "0:0-0:0" & !is.na(Sunday) &
Sunday != "0:0-0:0", 1, 0))

#Keep only total_hours and wkend_open
hours <- hours %>%
  select(wkend_open)

#Combine hours with business data
business_data <- cbind(business_data, hours)

#Generate business rating power, take natural logarithm to shrink range of values
business_data <- business_data %>%
  mutate(rating_power = log(review_count*stars, base = exp(1)))

#Business category
# Define the list of business categories (source:
https://blog.yelp.com/businesses/yelp_category_list/)
business_category <- c("Active Life", "Arts & Entertainment", "Automotive", "Beauty &
Spas", "Education", "Event Planning & Services", "Financial Services", "Food", "Health &
Medical", "Home Services", "Hotels & Travel", "Local Flavor", "Local Services", "Mass
Media", "Nightlife", "Pets", "Professional Services", "Public Services & Government",
"Real Estate", "Religious Organizations", "Restaurants", "Shopping")

# Create a new variable that maps the business category
business_data <- business_data %>%
  mutate(category = map_chr(categories, ~str_extract(., paste(business_category, collapse
= "|"))))

#Drop categories & hours
business_data <- business_data %>%
  select(-categories, -hours)

#--------------------------------------------------------------------------------#

### 2. TIP DATA
# Count the number of tips received by each business
tip_count <- table(tip_data$business_id)

# Convert the table to a data frame
tip_count_df <- as.data.frame(tip_count)
colnames(tip_count_df) <- c("business_id", "tip_count")

# Merge the tip_count with the business_data
business_data <- full_join(business_data, tip_count_df, by = "business_id")

# If a business_id does not have a tip review, replace NA with 0
business_data$tip_count[is.na(business_data$tip_count)] <- 0

#--------------------------------------------------------------------------------#

### 3. CHECKIN DATA
# number of check-ins recorded for each business
checkin_data$checkin_freq <- sapply(checkin_data$date, length)

#Drop checkin_data$date
checkin_data <- checkin_data %>%
  select(-date)

#Merge with business_data
business_data <- full_join(business_data, checkin_data, by = "business_id")
business_data$checkin_freq[is.na(business_data$checkin_freq)] <- 0

#--------------------------------------------------------------------------------#

### 4. REVIEW & USER DATA
#Keep only y variable (stars rating)
review_data_small <- review_data_small %>%
  select(review_id, user_id, business_id, stars, date)

#Keep relevant user info
user_data_small <- user_data_small %>%
  select(user_id, review_count, yelping_since, elite)

##Transform date variables to dttm format
review_data_small$date <- parse_datetime(review_data_small$date, format = "%Y-%m-%d
%H:%M:%S", na = c("", "NA"), locale = default_locale(), trim_ws = TRUE)
user_data_small$yelping_since <- parse_datetime(user_data_small$yelping_since, format =
"%Y-%m-%d %H:%M:%S", na = c("", "NA"), locale = default_locale(), trim_ws = TRUE)

## merge review data with user data
review_data <- left_join(review_data_small, user_data_small, by = "user_id")

# Create new variable to indicate number of years a user has been yelping
review_data$years_yelping <- as.numeric(difftime(review_data$date,
review_data$yelping_since, units = "weeks")) / 52.25
```

```
# Convert review date to year format
review_data$review_year <- as.numeric(format(as.Date(review_data$date), "%Y"))

# Create a function to check if review year is in elite years
is_elite <- function(elite, review_year) {
  # Check if elite_years is missing
  if (is.na(elite)) {
    return(0)
  }

  elite_years_vector <- as.numeric(str_split(elite, ",")[[1]])
  return(as.integer(review_year %in% elite_years_vector))
}

# Apply the function to each row
review_data$is_elite <- mapply(is_elite, review_data$elite, review_data$review_year)

review_data <- review_data %>%
  select(-elite, -review_year)

# Calculate the average stars for each business_id for elite and non-elite users
avg_stars <- review_data %>%
  group_by(business_id, is_elite) %>%
  summarise(avg_stars = mean(stars, na.rm = TRUE)) %>%
  spread(is_elite, avg_stars)

# Calculate the difference in averages
avg_stars$diff <- avg_stars$"1" - avg_stars$"0"

# Join the difference back to the original data frame
review_data <- review_data %>%
  left_join(avg_stars %>% select(business_id, diff), by = "business_id")

#--------------------------------------------------------------------------------#

### COMBINE ALL DATA
master_data <- inner_join(review_data, business_data, by = "business_id")

# User's average star rating given, by business category
cat_stars <- master_data %>%
  group_by(user_id, category) %>%
  summarise(cat_stars = mean(stars.x, na.rm = TRUE)) %>%
  pivot_wider(names_from = category, values_from = cat_stars)

# Convert cat_stars to long format
cat_stars_long <- cat_stars %>%
  pivot_longer(cols = -user_id, names_to = "category", values_to = "cat_stars")

# Merge master_data with cat_stars_long
master_data <- master_data %>%
  left_join(cat_stars_long, by = c("user_id", "category"))

#remove rows of missing x variables
master_data <- master_data[complete.cases(master_data), ]

#--------------------------------------------------------------------------------#

##Significant postcode
postal_code <- lm(stars.x ~ as.factor(postal_code), data = master_data)
summary_postcode <- summary(postal_code)

# Get the p-values
p_values <- summary_postcode$coefficients[, 4]

# Filter the postcodes based on the significance level
```

```
significant_postcode <- names(p_values)[p_values < 0.1]

# Print the significant postcodes
print(significant_postcode)

#Include new significant_postcode dummy
master_data$significant_postcode <- as.integer(master_data$postal_code %in%
significant_postcode)

##Significant city
cities <- lm(stars.x ~ as.factor(city), data = master_data)
summary_cities <- summary(cities)

# Get the p-values
p_values <- summary_cities$coefficients[, 4]

# Filter the cities based on the significance level
significant_cities <- names(p_values)[p_values < 0.1]

# Print the significant cities
print(significant_cities)

#Include new significant_postcode dummy
master_data$significant_city <- as.integer(master_data$city %in% significant_cities)

save(master_data, file = "master_data.Rda")

#---------------------------------------------------------------------------------#

## Basic correlation analysis

# Potential factors (check only numerical ones)
factors <- master_data[, c("stars.x", "review_count.x", "years_yelping", "is_elite",
"diff", "stars.y", "review_count.y", "wkend_open", "rating_power", "tip_count",
"checkin_freq", "cat_stars")]

correlations <- cor(factors, use = "pairwise.complete.obs")
# Get the variable names
vars <- rownames(correlations)

# Get the correlations with 'stars.x'
cors <- correlations[vars, "stars.x"]

# Create the data frame
correlation_data <- data.frame(Variable = vars, Correlation = cors)

# Plot the correlations
ggplot(correlation_data, aes(x = reorder(Variable, Correlation), y = Correlation)) +
  geom_bar(stat = "identity") +
  coord_flip() +
  labs(x = "Variable", y = "Correlation with stars.x", title = "Correlations with user's
star ratings")

#---------------------------------------------------------------------------------#

### MODELLING

## Final specification
formula <- as.formula("stars.x ~ review_count.x + years_yelping + is_elite + diff*is_elite
+ state + significant_city + significant_postcode + stars.y + review_count.y + wkend_open
+ rating_power + tip_count + checkin_freq + category + cat_stars")

set.seed(1)

# Split the data into training and test sets
```

```
sample_size <- 10000
test_indices <- sample(1:nrow(master_data), sample_size)
test_data <- master_data[test_indices, ]
train_data <- master_data[-test_indices, ]

# Prepare the data for glmnet
x <- model.matrix(formula, data = train_data)[,-1]  # remove intercept column
y <- train_data$stars.x

test_x <- model.matrix(formula, data = test_data)[,-1]  # remove intercept column

# x and test_x has unequal number of columns. Reconstruct test_x matrix to match training
data
test_x_full <- matrix(0, nrow = nrow(test_x), ncol = ncol(x))
colnames(test_x_full) <- colnames(x)
common_vars <- intersect(colnames(x), colnames(test_x))
test_x_full[, common_vars] <- test_x[, common_vars]


## Compare LASSO, Ridge and OLS

parameters <- c(seq(0.1, 2, 0.1) ,  seq(2, 5, 0.5) , seq(5, 25, 1))

lasso<-train(y = y,
              x = x,
              method = 'glmnet',
              tuneGrid = expand.grid(alpha = 1, lambda = parameters),
              metric =  "Rsquared"
)

ridge<-train(y = y,
              x = x,
              method = 'glmnet',
              tuneGrid = expand.grid(alpha = 0, lambda = parameters),
              metric =  "Rsquared"

)
linear<-train(y = y,
               x = x,
               method = 'lm',
               metric =  "Rsquared"
)

# Print the best parameters
print(paste0('Lasso best parameters: ' , lasso$finalModel$lambdaOpt))
print(paste0('Ridge best parameters: ' , ridge$finalModel$lambdaOpt))

predictions_lasso <- lasso %>% predict(test_x_full)
predictions_ridge <- ridge %>% predict(test_x_full)
predictions_lin <- linear %>% predict(test_x_full)

#rounds predicted star ratings to the nearest integer
rounded_lasso <- round(predictions_lasso)
rounded_ridge <- round(predictions_ridge)
rounded_lin <- round(predictions_lin) #rounds predicted star ratings to the nearest
integer

data.frame(
  Model = c("Ridge", "Lasso", "Linear"),
  Best_Lambda = c(ridge$finalModel$lambdaOpt, lasso$finalModel$lambdaOpt, NA),
  R_Squared = c(R2(predictions_ridge, test_data$stars.x), R2(predictions_lasso,
test_data$stars.x), R2(predictions_lin, test_data$stars.x)),
  RMSE = c(RMSE(predictions_ridge, test_data$stars.x), RMSE(predictions_lasso,
test_data$stars.x), RMSE(predictions_lin, test_data$stars.x)),
  Normalised_RMSE = c(RMSE(predictions_ridge, test_data$stars.x)/4,
RMSE(predictions_lasso, test_data$stars.x)/4, RMSE(predictions_lin, test_data$stars.x)/4),
  RMSE_round = c(RMSE(rounded_ridge, test_data$stars.x), RMSE(rounded_lasso,
test_data$stars.x), RMSE(rounded_lin, test_data$stars.x)),
  Normalised_RMSE_round = c(RMSE(rounded_ridge, test_data$stars.x)/4, RMSE(rounded_lasso,
test_data$stars.x)/4, RMSE(rounded_lin, test_data$stars.x)/4)
)
```