Appendix:

Table 1: Short description of the news data set predictors

0. url:	URL of the article
1. timedelta:	Days between the article publication and dataset acquisition
2. n_tokens_title:	Number of words in the title
3. n_tokens_content:	Number of words in the content
4. n_unique_tokens:	Rate of unique words in the content
5. n_non_stop_words:	Rate of non-stop words in the content
6. n_non_stop_unique_tokens:	Rate of unique non-stop words in the content
7. num_hrefs:	Number of links
8. num_self_hrefs:	Number of links to other articles published by Mashable
9. num_imgs:	Number of images
10. num_videos:	Number of videos
11. average_token_length:	Average length of the words in the content
12. num_keywords:	Number of keywords in the metadata
13. data_channel_is_lifestyle:	Is data channel 'Lifestyle'?
14. data_channel_is_entertainment:	Is data channel 'Entertainment'?
15. data_channel_is_bus:	Is data channel 'Business'?
16. data_channel_is_socmed:	Is data channel 'Social Media'?
17. data_channel_is_tech:	Is data channel 'Tech'?
18. data_channel_is_world:	Is data channel 'World'?
19. kw_min_min:	Worst keyword (min. shares)
20. kw_max_min:	Worst keyword (max. shares)
21. kw_avg_min:	Worst keyword (avg. shares)
22. kw_min_max:	Best keyword (min. shares)
23. kw_max_max:	Best keyword (max. shares)
24. kw_avg_max:	Best keyword (avg. shares)
25. kw_min_avg:	Avg. keyword (min. shares)
26. kw_max_avg:	Avg. keyword (max. shares)
27. kw_avg_avg:	Avg. keyword (avg. shares)
28. self_reference_min_shares:	Min. shares of referenced articles in Mashable
29. self_reference_max_shares:	Max. shares of referenced articles in Mashable
30. self_reference_avg_sharess:	Avg. shares of referenced articles in Mashable
31. weekday_is_monday:	Was the article published on a Monday?
32. weekday_is_tuesday:	Was the article published on a Tuesday?
33. weekday_is_wednesday:	Was the article published on a Wednesday?
34. weekday_is_thursday:	Was the article published on a Thursday?
35. weekday_is_friday:	Was the article published on a Friday?

36. weekday_is_saturday:	Was the article published on a Saturday?
37. weekday_is_sunday:	Was the article published on a Sunday?
38. is_weekend:	Was the article published on the weekend?
39. LDA_00:	Closeness to LDA topic 0
40. LDA_01:	Closeness to LDA topic 1
41. LDA_02:	Closeness to LDA topic 2
42. LDA_03:	Closeness to LDA topic 3
43. LDA_04:	Closeness to LDA topic 4
44. global_subjectivity:	Text subjectivity
45. global_sentiment_polarity:	Text sentiment polarity
46. global_rate_positive_words:	Rate of positive words in the content
47. global_rate_negative_words:	Rate of negative words in the content
48. rate_positive_words:	Rate of positive words among non-neutral tokens
49. rate_negative_words:	Rate of negative words among non-neutral tokens
50. avg_positive_polarity:	Avg. polarity of positive words
51. min_positive_polarity:	Min. polarity of positive words
52. max_positive_polarity:	Max. polarity of positive words
53. avg_negative_polarity:	Avg. polarity of negative words
54. min_negative_polarity:	Min. polarity of negative words
55. max_negative_polarity:	Max. polarity of negative words
56. title_subjectivity:	Title subjectivity
57. title_sentiment_polarity:	Title polarity
58. abs_title_subjectivity:	Absolute subjectivity level
59. abs_title_sentiment_polarity:	Absolute polarity level
60. shares:	Number of shares (target)

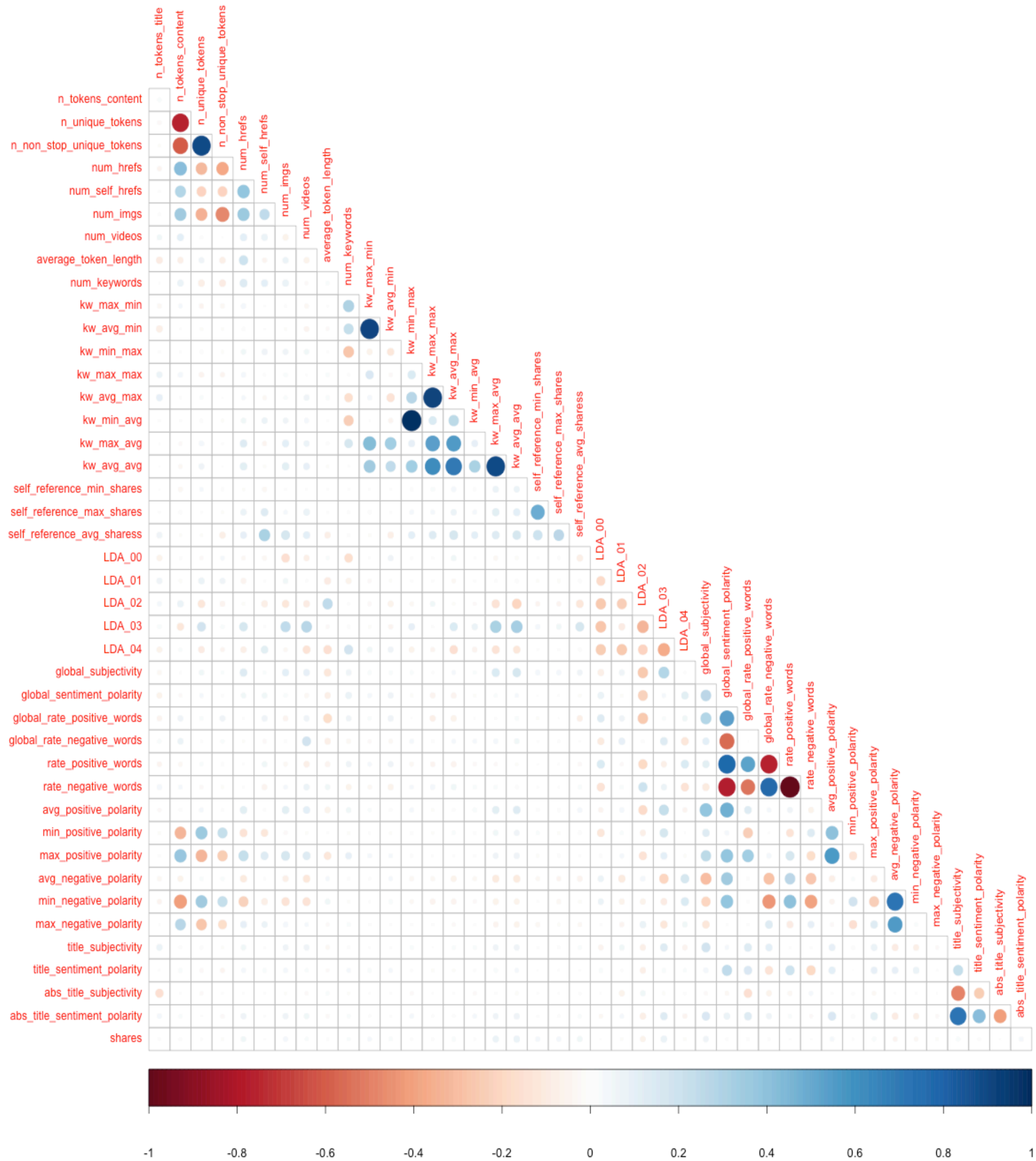

Figure 1: Graphical representation of Continuous predictors' correlation matrix

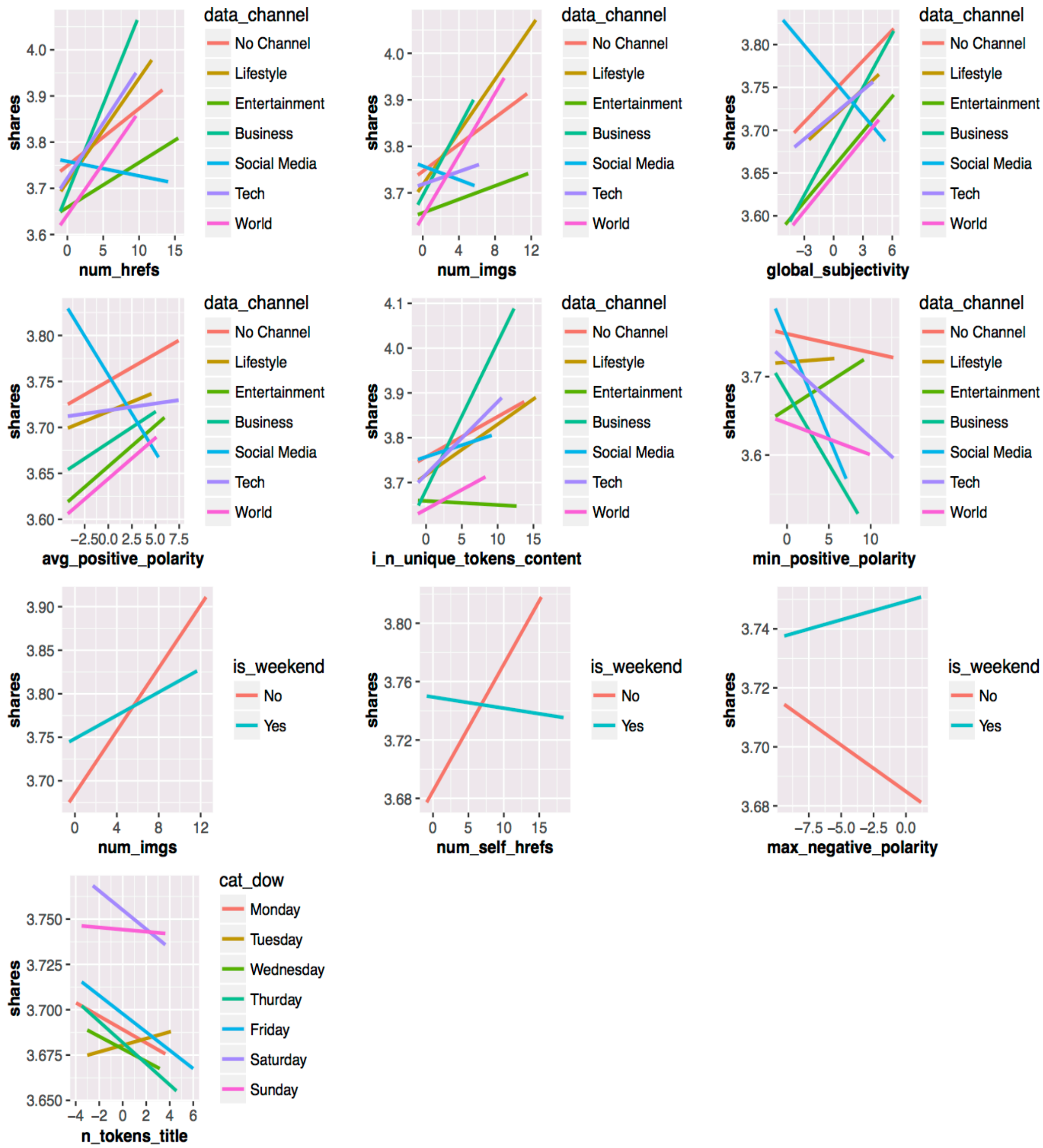

Figure 2: Graphical representation of Interaction between continuous variables and categorical variables considering response variable

R Code Sections

Section 1: Loading dataset
Section 2: Data Cleaning
Section 3: Stepwise Regression Model
Section 4: LASSO and RIDGE (Regularization)
Section 5: Weighted Regression
Section 6: Bootstrap
Section 7: R Plot Scripts

Section 1: Loading dataset

```
library(dplyr)
library(car)
library(caret)
library(glmnet)
library(ggplot2)
library(grid)
library(gridExtra)

setwd("/Users/Darshan/Documents/Online_News_Popularity")

# Train - Test data split
news <- read.csv("OnlineNewsPopularity.csv", header = TRUE)

set.seed(10)

# shuffle the data set
news <- news[sample(1:nrow(news)),]

# 20 % of data will be separated for testing
n_test_samples <- round(nrow(news) * 0.20)

news_test <- news[(1:n_test_samples),]
news_train <- news[((1+n_test_samples):nrow(news)),]

write.csv("Train.csv", row.names = FALSE, x = news_train)
write.csv("Test.csv", row.names = FALSE, x = news_test)
```

Section 2: Data Cleaning

```
# This function cleans the train dataset
data_cleaning <- function(news){

 # non-predictor
 news$timedelta <- NULL

 # Removing instances which don't have any text content in it.
 news <- filter(news, n_tokens_content != 0)
```

```r
  news$n_non_stop_words <- NULL # Constant predictor
  news$kw_min_min <- NULL # More than 50% instances contain -1 value
  news <- filter(news, n_unique_tokens <= 1) # Outlier value greater than 1

  news <- filter(news, kw_min_avg >= 0, kw_avg_min>=0)
  # Outlier value less than 1

  # Log transformation becuase there are high number of outliers
  news$LDA_00 <- log(news$LDA_00 + 1)
  news$LDA_01 <- log(news$LDA_01 + 1)
  news$LDA_02 <- log(news$LDA_02 + 1)
  news$LDA_03 <- log(news$LDA_03 + 1)
  news$LDA_04 <- log(news$LDA_04 + 1)

  news$self_reference_avg_sharess <- log(news$self_reference_avg_sharess + 1)

  news$kw_max_min <- log(news$kw_max_min + 1)
  news$kw_avg_min <- log(news$kw_avg_min + 1)

  news$kw_min_avg <- log(news$kw_min_avg + 1)
  news$kw_min_max <- log(news$kw_min_max + 1)
  news$kw_max_avg <- log(news$kw_max_avg + 1)
  news$kw_avg_avg <- log(news$kw_avg_avg + 1)

  news$kw_avg_max <- log(news$kw_avg_max + 1)
  news$kw_max_max <- log(news$kw_max_max + 1)

  return(news)

}

# This function handle the multi-collinearity in the train dataset.
correlation_cleaning <- function(news){

  news$rate_negative_words <- NULL
  # n_non_stop_unique_tokens and n_unique_tokens have correlation of 0.887
  #  n_non_stop_unique_tokens is removed from the analysis as both the predictors
  # are semantically similar
  news$n_non_stop_unique_tokens <- NULL

  # self_reference_min_shares and self_reference_max_shares has high corelation with
  # self_reference_avg_sharess
  news$self_reference_min_shares <- NULL
  news$self_reference_max_shares <- NULL

  news$i_n_unique_tokens_content <- news$n_unique_tokens + news$n_tokens_content
  # 0.751 colinearity between n_unique_tokens and n_tokens_content
  news$n_unique_tokens <- NULL
```

```
news$n_tokens_content <- NULL

news$i_title_sub_sent_polarity <- (news$title_subjectivity +
                        news$abs_title_sentiment_polarity) / 2.0
# 0.71 colinearity between title_subjectivity and abs_title_sentiment_polarity
news$title_subjectivity <- NULL
news$abs_title_sentiment_polarity <- NULL

# 0.719 colinearity between min_negative_polarity and avg_negative_polarity
news$i_min_avg_negative_pol <- (news$min_negative_polarity +
                        news$avg_negative_polarity) / 2.0
news$min_negative_polarity <- NULL
news$avg_negative_polarity <- NULL

# 0.779 colinearity between rate_positive_words and global_sentiment_polarity
news$i_rate_pos_gsent_polarity <- (news$rate_positive_words *
                        news$global_sentiment_polarity)
news$rate_positive_words <- NULL
news$global_sentiment_polarity <- NULL

#kw_max_min and kw_avg_min have correlation of 0.901
news$i_kw_max_avg_min <- (news$kw_max_min + news$kw_avg_min) / 2.0
#kw_max_avg and kw_avg_avg have correlation of 0.899
news$i_kw_max_avg_avg <- (news$kw_max_avg + news$kw_avg_avg) / 2.0

# High collinearity after applying log transformation on kw_min_avg and kw_min_max
# Log transformation has improved the r-squared value
news$kw_min_max<- NULL

# High collinearity after applying log transformation on kw_avg_max and kw_max_max
# Log transformation has improved the r-squared value
news$i_kw_avg_max_max <- (news$kw_avg_max + news$kw_max_max) / 2.0
news$kw_avg_max <- NULL
news$kw_max_max <- NULL

news$kw_max_min <- NULL
news$kw_avg_min <- NULL
news$kw_max_avg <- NULL
news$kw_avg_avg <- NULL

# After trying different interactions between the predictors,
# correlation did not decrease significantly, so
# self_reference_min_shares and self_reference_max_shares
# predictors are both removed.
news$self_reference_min_shares <- NULL
news$self_reference_max_shares <- NULL

return(news)
```

```r
}

# This function applies the Box-Cox transformation on responce variable
target_transformation <- function(news) {

  p <- powerTransform(news$shares)
  shares_transformed <- bcPower(news$shares, p$lambda)
  news$shares <- shares_transformed

  return(list("news"=news, "lambda"=p$lambda))
}

# This function returns the actual value of the responce variable
# from the Box-Cox transformation
target_inverse <- function(shares, lambda) {
  if (lambda == 0) {
    shares <- exp(shares)
  }
  else {
    shares <- (lambda*shares + 1)^(1/lambda)
  }

  return(shares)
}

# This funciton normalize continuous variables of the train dataset
normalization <- function(news_train){

  # All Column names
  column_names <- names(news_train)

  # Column names which needs to be ignored due to categorical and target feature
  ignored_column_names <- c("url", "timedelta", "data_channel_is_lifestyle",
                      "data_channel_is_entertainment", "data_channel_is_bus",
                      "data_channel_is_world", "data_channel_is_socmed",
                      "data_channel_is_tech", "weekday_is_monday",
                      "weekday_is_tuesday",
                      "weekday_is_wednesday", "weekday_is_thursday",
                      "weekday_is_friday",
                      "weekday_is_saturday", "weekday_is_sunday", "is_weekend",
                      "shares")

  needed_columns <- setdiff(column_names,ignored_column_names)

  # Normalized Train Data
  #news_train_norm <- news_train %>% mutate_each_(funs(scale),vars=needed_columns)
```

```
  # Saving standard deviation of the columns which are normalized
  sd_values <- Map(sd, news_train[,needed_columns])

  # Saving mean of the columns which are normalized
  mean_values <- Map(mean, news_train[,needed_columns])

  news_train[,needed_columns] <- (news_train[,needed_columns] - mean_values) / sd_values

  return(list("sd_values"=sd_values, "mean_values"=mean_values, "news_train"=news_train))

}

# This funciton normalize continuous variables of the test datset
apply_normalization <- function(news, means, sds) {
  # All Column names
  column_names <- names(news_train)

  # Column names which needs to be ignored due to categorical and target feature

  ignored_column_names <- c("url", "timedelta", "data_channel_is_lifestyle",
                          "data_channel_is_entertainment", "data_channel_is_bus",
                          "data_channel_is_world", "data_channel_is_socmed",
                          "data_channel_is_tech", "weekday_is_monday",
                          "weekday_is_tuesday",
                          "weekday_is_wednesday", "weekday_is_thursday",
                          "weekday_is_friday",
                          "weekday_is_saturday", "weekday_is_sunday", "is_weekend",
                          "shares")

  needed_columns <- setdiff(column_names,ignored_column_names)

  news[,needed_columns] <- (news[,needed_columns] - means) / sds

  return(news)
}

# This function creates the factor/single categorical variable by combining
# multiple/one hot encoded variables
cat_encoding <- function(news){

  dow_cols = c("weekday_is_monday", "weekday_is_tuesday", "weekday_is_wednesday",
            "weekday_is_thursday", "weekday_is_friday", "weekday_is_saturday",
            "weekday_is_sunday")

  news$cat_dow <- 0

  for (dow in dow_cols) {
    dow_idx = which(news[,dow] == 1)
```

```r
    #print(dow_idx)
    news[dow_idx,"cat_dow"] <- which(dow_cols==dow)
  }

  news$cat_dow <- as.factor(news$cat_dow)

  data_channel_cols = c("data_channel_is_lifestyle", "data_channel_is_entertainment",
                        "data_channel_is_bus", "data_channel_is_socmed",
                        "data_channel_is_tech",
                        "data_channel_is_world")

  news$data_channel <- 0

  for (channel in data_channel_cols) {
    channel_idx <- which(news[,channel] == 1)
    news[channel_idx,"data_channel"] <- which(data_channel_cols==channel)
  }

  news$data_channel <- as.factor(news$data_channel)

  news$is_weekend <- as.factor(news$is_weekend)

  return(news)

}

OUTLIERS_HIGH_CUTOFF = 0.1
OUTLIERS_LOW_CUTOFF = 0.05
outliers_removal <- function(news) {
  # sort by shares
  sorted_news <- news[order(news$shares),]

  num_rows <- nrow(news)
  # remove lower tail
  cut_low_point <- as.integer(OUTLIERS_LOW_CUTOFF*num_rows)
  cut_high_point <- as.integer((1-OUTLIERS_HIGH_CUTOFF)*num_rows)
  sorted_news <- sorted_news[cut_low_point:cut_high_point, ]
  news <- sorted_news[sample(nrow(sorted_news)),]
  return(sorted_news)
}

# This function removes the outlier from the dataset based upon the
# cook's distance
cook_outliers_removal <- function(news){

  cutoff <- 4/nrow(news)
  model <- lm(shares ~ ., data=news)
  infl <- lm.influence(model, do.coef = FALSE)
```

```
  cooks.distance <- cooks.distance(model, infl = infl,
                                   res = weighted.residuals(model),
                                   sd = sqrt(deviance(model)/df.residual(model)),
                                   hat = infl$hat)

  index <- cooks.distance <= cutoff
  news <- news[index,]

  return(news)

}

# This function loads the train data set and applies the
# data cleaning operation to it.
load_processed_train_data <- function(outliers.removed=FALSE,
                                      one.hot.encoding.remove=TRUE){

  news <- read.csv("Train.csv", header = TRUE)

  news <- data_cleaning(news)
  news <- correlation_cleaning(news)

  obj <- normalization(news)
  news <- obj$news

  news <- cat_encoding(news)

  url <- news$url
  news$url <- NULL

  if(one.hot.encoding){

    categorical_var <- c("data_channel_is_lifestyle",
                     "data_channel_is_entertainment", "data_channel_is_bus",
                     "data_channel_is_world", "data_channel_is_socmed",
                     "data_channel_is_tech", "weekday_is_monday", "weekday_is_tuesday",
                     "weekday_is_wednesday", "weekday_is_thursday", "weekday_is_friday",
                     "weekday_is_saturday", "weekday_is_sunday")

    news_with_cat <- subset(news, select = categorical_var)
    news <- subset(news, select = setdiff(names(news),categorical_var))
  }

  if(outliers.removed){
    news <- cook_outliers_removal(news)
  }
```

```
  return(news)
}
```

Section 3: Stepwise Regression Model

```
set.seed(464)

news <- load_processed_train_data()

K <- 10
# 10 - fold cross validation
folds <- createFolds(news$shares, k = K, list=TRUE, returnTrain=TRUE)

models <- list()
rmses <- c()
R2s <- c()

for (i in 1:K) {

  news_train <- news[folds[[i]],]
  news_val <- news[-folds[[i]],]

  null=lm(shares~1, data=news_train)
  full=lm(shares~., data=news_train)

  model <- step(null, scope=list(lower=null, upper=full), direction="both", trace=0)
  #model <- step(full, direction="backward", trace=0)

  pred <- predict(model, news_val)

  #pred <- target_inverse(pred, lamda)
  #shares_val <- target_inverse(news_val$shares, lamda)
  #mse <- sum((pred - shares_val)**2) / nrow(news_val)

  mse <- sum((pred - news_val$shares)**2) / nrow(news_val)
  rmses <- append(rmses, sqrt(mse))

  R2s <- append(R2s, summary(model)$adj.r.squared)

  models[[i]] <- model

}

# Displaying which variables are selected in the each fold
unique_coef <- c()

for(i in 1:length(models)){
  model_coef <- names(models[[i]]$coefficients)
```

```
  unique_coef <- unique(c(model_coef, unique_coef))
}

model_variables <- data.frame(matrix(NA,nrow=length(unique_coef),ncol=length(models)+1))
model_variables$X1 <- unique_coef

for(i in 1:length(models)){

  model_coef <- names(models[[i]]$coefficients)
  tf_coef <- unique_coef %in% model_coef
  var <- paste("X", toString(i+1), sep = "")
  model_variables[var] <- tf_coef

}
```

Section 4: LASSO and RIDGE (Regularization)

```
set.seed(464)

# run grid search with cross validation to select best values for lambda and alpha in elastic net
select_model <- function(news, t_lambda) {
  K = 10
  # alpha = 0 -> Ridge; alpha = 1 -> Lasso
  alphas = c(0,1)
  lambdas = c(1e-05, 1e-04, 1e-03, 1e-02, 0.1, 1.,10.)

  folds <- createFolds(news$shares, k = K, list=TRUE, returnTrain=TRUE)
  # for each combination of parameters
  for (alpha in alphas) {
   for (lambda in lambdas) {
     rmses <- c()
     R2s <- c()

     # for each fold
     for (i in 1:K) {
       news_train <- news[folds[[i]],]
       news_val <- news[-folds[[i]],]

       X_train <- data.matrix(subset(news_train,select=-shares))
       y_train <- data.matrix(news_train$shares)
       X_val <- data.matrix(subset(news_val,select=-shares))
       y_val <- data.matrix(news_val$shares)

       model <- glmnet(X_train, y_train, family="gaussian", alpha=alpha, standardize=TRUE,
                lambda=lambda, nlambda=1)

       pred_train <- predict(model, newx=X_train, s=lambda)
       shares_train <- y_train
```

```r
      # calculate R^2 in the fitted data
      ssto <- sum((shares_train - mean(shares_train))**2)
      sse <- sum((pred_train - shares_train)**2)
      R2 <- 1 - (sse/ssto)

      R2s <- append(R2s, R2)

      pred <- predict(model, newx=X_val, s=lambda)
      shares_val <- y_val

      sse <- sum((pred - shares_val)**2)
      rmse <- sqrt(sse / nrow(news_val))

      rmses <- append(rmses,rmse)
    }
    mrmse= mean(rmses)
    srmse= sd(rmses)
    mR2 = mean(R2s)
    cat(sprintf("alpha = %f, lambda = %f, avg rmse = %f, sd rmse = %f, avg R-2 = %f\n",
            alpha, lambda, mrmse,srmse,mR2))
  }
 }

}

news <- load_processed_train_data()
select_model(news, t_lambda)
```

Section 5: Weighted Regression

```r
ncvTest(lm(shares ~ .,data=news))
m.unweighted <- lm(shares ~ ., data=news)

# Learing weights of each data point
w <- predict(lm(abs(m.unweighted$res) ~ predict(m.unweighted, data=news)), data=news)
# First Approach, updating responce variable based upon weights
#w <- (w - min(w))/(max(w) - min(w))
#news$shares <- news$shares * w

K <- 10
# Third Approach; Learning from the weights
model <- lm(formula = shares ~ ., data = news, weights = 1/(w^2))

folds <- createFolds(news$shares, k = K, list=TRUE, returnTrain=TRUE)

models <- list()
rmses <- c()
```

```
R2s <- c()
for (i in 1:K) {

  news_train <- news[folds[[i]],]
  news_val <- news[-folds[[i]],]
  #w <- w[folds[[i]]]

  m.unweighted <- lm(shares ~ ., data=news_train)
  w <- predict(lm(abs(m.unweighted$res) ~ predict(m.unweighted, data=news_train)), data=news_train)
  # Second Approach, updating responce variable based upon weights and fold
  #w <- (w - min(w))/(max(w) - min(w))
  #news_train$shares <- news_train$shares * w

  null=lm(shares~1, data=news_train)
  full=lm(shares~., data=news_train)

  model <- lm(formula = shares ~ ., data = news_train, weights = 1/(w^2))
  #model <- step(null, scope=list(lower=null, upper=full), direction="forward", trace=0)

  pred <- predict(model, news_val)

  #pred <- target_inverse(pred, lamda)
  #shares_val <- target_inverse(news_val$shares, lamda)
  #mse <- sum((pred - shares_val)**2) / nrow(news_val)

  mse <- sum((pred - news_val$shares)**2) / nrow(news_val)
  rmses <- append(rmses, sqrt(mse))

  R2s <- append(R2s, summary(model)$adj.r.squared)

  models[[i]] <- model

}
```

Section 6: Bootstrap

```
set.seed(464)
news <- load_processed_train_data()
B = 300
bootstrap <- function(formula, data) {
  n_rows <- nrow(data)

  models <- vector(mode="list", length=B)
  for (i in 1:B) {
    # sample the same number of points with replacement
    boot_idx <- sample(n_rows, n_rows, replace = TRUE)
    boot_data <- data[boot_idx, ]
```

```r
    m <- lm(formula, data=boot_data)

    models[[i]] <- m
  }

  return(models)
}


# stepwise selection (with outliers)
predictors <- c("data_channel", "cat_dow", "i_kw_max_avg_avg",
               "self_reference_avg_sharess", "i_kw_avg_max_max",
               "num_hrefs", "global_subjectivity", "LDA_00",
               "LDA_01", "LDA_02", "num_self_hrefs",
               "i_n_unique_tokens_content", "i_title_sub_sent_polarity",
               "abs_title_subjectivity", "n_tokens_title", "min_positive_polarity",
               "num_imgs", "average_token_length", "title_sentiment_polarity",
               "i_min_avg_negative_pol")

# # # stepwise selection (without outliers)
# predictors <- c("num_hrefs", "num_self_hrefs", "num_imgs",
#   "self_reference_avg_sharess", "LDA_00", "LDA_02", "global_subjectivity",
#   "global_rate_positive_words", "global_rate_negative_words", "min_positive_polarity",
#   "max_negative_polarity", "title_sentiment_polarity", "abs_title_subjectivity",
#   "i_n_unique_tokens_content", "i_rate_pos_gsent_polarity", "i_kw_max_avg_avg",
#   "i_kw_avg_max_max", "cat_dow", "data_channel", "i_title_sub_sent_polarity")


formula <- as.formula(paste("shares~", paste(predictors,collapse="+")))

# number of coefficients in the model
N_COEF <- 31

# get the coefficients values from each model
coef <- matrix(nrow = B, ncol=N_COEF)
models <- bootstrap(formula, news)
for (i in 1:length(models)) {
  for (j in 2:N_COEF) {
    coef[i,j] <- coef(models[[i]])[[j]]
  }
}

# train a model on the full dataset
full_model <- lm(formula, data=news)
full_coef <- vector(mode="list", length=N_COEF)
predictor_names <- names(full_model$coefficients)[2:N_COEF]

# get the coefficients of the full model
```

```r
for (i in 2:N_COEF) {
  full_coef[[i]] <- coef(full_model)[[i]]
}

# calculate coefficients confidence intervals
coef_max <- vector(mode="list", length=N_COEF)
coef_min <- vector(mode="list", length=N_COEF)
for (i in 2:N_COEF) {
  b_star_upper <- qnorm(0.975, mean=mean(coef[,i]), sd=sd(coef[,i]))
  b_star_lower <- qnorm(0.025, mean=mean(coef[,i]), sd=sd(coef[,i]))

  d1 <- full_coef[[i]] - b_star_upper
  d2 <- b_star_lower - full_coef[[i]]

  coef_max[[i]] <-  full_coef[[i]] - d2
  coef_min[[i]] <- full_coef[[i]] + d1

  cat(sprintf("predictor: %s, lower_value = %f, upper_value = %f\n",
            predictor_names[i], coef_min[[i]], coef_max[[i]]))
}

# plot the coefficient and their confidence interval
results = data.frame(name=predictor_names, coef=unlist(full_coef), max=unlist(coef_max), min=unlist(
coef_min))

ggplot(results, aes(x = name, y = coef)) +
  geom_point(size = 1) +
  labs(x = "Predictor", y = "Estimated coefficient") +
  geom_errorbar(aes(ymax = max, ymin = min),width=0.1) +
  theme(axis.text.x = element_text(angle = 90, hjust = 1, size=10, face="bold"))

# prediction for all the models
pred <- matrix(nrow = nrow(news), ncol=B)
for (i in 1:length(models)) {
  m <- models[[i]]
  pred[,i] <- predict(m, subset(news,select=-shares))
}

sse <- sum((rowMeans(pred) - news$shares)**2)
rmse <- sqrt(sse / nrow(news))
```

Section 7: R Plot Scripts

```r
news <- load_processed_train_data()

news_wo_outlier <- cook_outliers_removal(news)

model <- lm(shares ~ ., data=news)
```

```
news$res <- abs(model$residuals)
news$pre <- predict(model, data=news)

model <- lm(shares ~ ., data=news_wo_outlier)

news_wo_outlier$res <- abs(model$residuals)
news_wo_outlier$pre <- predict(model, data=news_wo_outlier)

p1 <- ggplot(aes(x=pre,y=res) ,data=news)  +  geom_point() + xlab("Predicted Number of Shares (With O
utliers)") + ylab("abs(Residual)") + stat_binhex(bins = 75)  + geom_smooth(color = "red") + theme(axis.ti
tle=element_text(size=9,face="bold"))

p2 <- ggplot(aes(x=pre,y=res) ,data=news_wo_outlier)  +  geom_point() + xlab("Predicted Number of Sh
ares (Without Outliers)") + ylab("abs(Residual)") + stat_binhex(bins = 75)  + geom_smooth(color = "red"
) + theme(axis.title=element_text(size=9,face="bold"))

grid.arrange(p1, p2, ncol = 2, top = "Residual vs Predicted value of Shares")

news <- load_processed_train_data()

model <- lm(shares ~ data_channel + cat_dow + i_kw_max_avg_avg +
          self_reference_avg_sharess + i_kw_avg_max_max +
          num_hrefs + global_subjectivity + LDA_00 +  LDA_01 +
          LDA_02 + num_self_hrefs + i_n_unique_tokens_content +
           i_title_sub_sent_polarity + abs_title_subjectivity +
          n_tokens_title +  min_positive_polarity +
          num_imgs + average_token_length + title_sentiment_polarity +
          i_min_avg_negative_pol, data=news)

news$res <- abs(model$residuals)
news$pre <- predict(model, data=news)
y <- quantile(news$res, c(0.25, 0.75))
x <- qnorm(c(0.25, 0.75))
slope <- diff(y)/diff(x)
int <- y[1L] - slope * x[1L]

p1 <- ggplot(news, aes(sample=res)) + stat_qq() + geom_abline(slope = slope, intercept = int)  + ylab("St
epwise Regression model residuals (With Outlier)") + theme(axis.title=element_text(size=9,face="bold")
)

news_wo_outlier <- cook_outliers_removal(news)

model <- lm(shares ~ num_hrefs +  num_self_hrefs + num_imgs +
           self_reference_avg_sharess + LDA_00 + LDA_02 +
          global_subjectivity + global_rate_positive_words + global_rate_negative_words +
          min_positive_polarity + max_negative_polarity + title_sentiment_polarity +
          abs_title_subjectivity + i_n_unique_tokens_content +
```

```
        i_rate_pos_gsent_polarity + i_kw_max_avg_avg + i_kw_avg_max_max +
        cat_dow + data_channel +
        i_title_sub_sent_polarity, data=news_wo_outlier)

news_wo_outlier$res <- abs(model$residuals)
news_wo_outlier$pre <- predict(model, data=news)

y <- quantile(news_wo_outlier$res, c(0.25, 0.75))
x <- qnorm(c(0.25, 0.75))
slope <- diff(y)/diff(x)
int <- y[1L] - slope * x[1L]

p2 <- ggplot(news_wo_outlier, aes(sample=res)) + stat_qq() +geom_abline(slope = slope, intercept = int
) + ylab("Stepwise Regression model residuals (Without Outlier)") + theme(axis.title=element_text(size=
9,face="bold"))

grid.arrange(p1, p2, ncol = 2, top = "Residual QQ Plots")
```

Statement of Contributions

Darshan Patel mainly worked on building the Stepwise and Weighted least square regression models. He programmed various variable selection methods in R and compared their results. He also wrote R scripts to visualize Multi-collinearity, Interaction Terms and Residual plots. Along with that he assisted Gabriel in building LASSO, Adaptive LASSO, RIDGE and Bootstrap and Data cleaning process.

Gabriel Bakiewicz mainly worked on building the Regularization and Bootstrap models. He programmed LASSO, Adaptive LASSO, RIDGE and Bootstrap in R and build scripts to visualize the confidence interval of the predictor estimation and compare the different models. Along with that he assisted Darshan in building Stepwise and Weighted least square regression models and Data cleaning process.