

# Boeing Data Science Challenge Problem

Hyunjoon Rhee

4/14/23

- Introduction
  - Project Goal
  - What are the variables?
- Methods
  - Data Cleaning
  - Data Preprocessing
  - Training and Testing

## Introduction

## Project Goal

This dataset contains information on used cars previously sold.

Predict: Vehicle Trim & Dealer listing price

How? Create one model -vehicle trim another model - dealer listing price

## What are the variables?

```
ucd = read.csv("Training_DataSet.csv") #ucd = usedcardata

skim(ucd)
```

Data summary

Name	ucd
Number of rows	6298
Number of columns	29
Column type frequency:	
character	19
logical	2
numeric	8
Group variables	
None	

Variable type: character

skim_variable	n_missing	complete_rate	min	max	empty	n_unique	whitespace
SellerCity	0	1	3	28	0	1318	0
SellerListSrc	0	1	0	27	2	9	0
SellerName	0	1	3	73	0	2452	0

SellerState	0	1	2	2	0	50	0
VehBodystyle	0	1	3	3	0	1	0
VehColorExt	0	1	0	41	73	170	0
VehColorInt	0	1	0	50	728	107	0
VehDriveTrain	0	1	0	58	401	21	0
VehEngine	0	1	0	64	361	97	0
VehFeats	0	1	0	2006	275	844	0
VehFuel	0	1	0	13	2	5	0
VehHistory	0	1	0	111	201	34	0
VehMake	0	1	4	8	0	2	0
VehModel	0	1	3	14	0	2	0
VehPriceLabel	0	1	0	10	285	4	0
VehSellerNotes	0	1	0	4103	243	4921	0
VehType	0	1	4	4	0	1	0
VehTransmission	0	1	0	29	197	34	0
Vehicle_Trim	0	1	0	32	405	30	0

Variable type: logical

skim_variable	n_missing	complete_rate	mean	count
SellerIsPriv	0	1	0.00	FAL: 6284, TRU: 14
VehCertified	0	1	0.23	FAL: 4879, TRU: 1419

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
ListingID	0	1.00	4318129.97	2486030.99	3287.00	2178112	4298122.00	6488249.00	8620012.00	
SellerRating	0	1.00	4.14	1.19	0.00	4	4.60	4.80	5.00	
SellerRevCnt	0	1.00	434.57	1274.26	0.00	28	126.00	401.00	14635.00	
SellerZip	2	1.00	45234.21	20380.48	1105.00	28806	46410.00	60126.00	99654.00	
VehListdays	2	1.00	56.14	68.30	0.29	13	33.46	74.14	820.68	
VehMileage	2	1.00	26369.36	13036.57	0.00	16835	26181.00	36468.50	83037.00	
VehYear	0	1.00	2016.79	1.21	2015.00	2015	2017.00	2018.00	2019.00	
Dealer_Listing_Price	52	0.99	32265.05	7538.34	18289.00	26900	31455.50	35991.00	89500.00	

```
ucd_validation = read.csv("Test_Dataset.csv")
skim(ucd_validation)
```

Data summary

Name	ucd_validation
Number of rows	1000

Number of columns	27
Column type frequency:	
character	18
logical	2
numeric	7
Group variables	
None	


Variable type: character

skim_variable	n_missing	complete_rate	min	max	empty	n_unique	whitespace
SellerCity	0	1	4	28	0	573	0
SellerListSrc	0	1	16	26	0	7	0
SellerName	0	1	4	64	0	706	0
SellerState	0	1	2	2	0	44	0
VehBodystyle	0	1	3	3	0	1	0
VehColorExt	0	1	0	34	7	91	0
VehColorInt	0	1	0	50	108	53	0
VehDriveTrain	0	1	0	17	64	16	0
VehEngine	0	1	0	64	58	55	0
VehFeats	0	1	0	2006	37	247	0
VehFuel	0	1	6	13	0	3	0
VehHistory	0	1	0	110	27	22	0
VehMake	0	1	4	8	0	2	0
VehModel	0	1	3	14	0	2	0
VehPriceLabel	0	1	0	10	38	4	0
VehSellerNotes	0	1	0	4044	41	852	0
VehType	0	1	4	4	0	1	0
VehTransmission	0	1	0	27	27	22	0

Variable type: logical

skim_variable	n_missing	complete_rate	mean	count
SellerIsPriv	0	1	0.00	FAL: 998, TRU: 2
VehCertified	0	1	0.23	FAL: 771, TRU: 229

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
ListingID	0	1	9303279.32	392260.42	8622015.00	8984289.75	9297767.50	9644453.25	9999562.0	

SellerRating	0	1	4.07	1.26	0.00	3.80	4.60	4.80	5.0	
SellerRevCnt	0	1	488.27	1513.65	0.00	28.00	130.00	386.00	15822.0	
SellerZip	0	1	44297.52	20225.28	1581.00	27948.50	46046.00	57191.50	99518.0	
VehListdays	0	1	53.47	60.04	0.31	12.66	32.49	74.43	430.9	
VehMileage	1	1	25679.65	12583.41	0.00	16620.50	25167.00	35534.50	58478.0	
VehYear	0	1	2016.79	1.22	2015.00	2015.00	2017.00	2018.00	2019.0	

## Methods

## Data Cleaning

### Remove unnecessary columns

Before removing NA, select columns that can be removed

```
ucd <- subset(ucd, select = -c(VehType,VehBodystyle,ListingID,SellerCity,SellerState,SellerName,VehSellerNotes,VehColorInt))

# Testing value
list_ID <- ucd_validation[,1]
list_ID <- as.data.frame(list_ID)
ucd_validation <- subset(ucd_validation, select = -c(VehType,VehBodystyle,ListingID,SellerCity,SellerState,SellerName,VehSellerNotes,VehColorInt))
```

- VehType, VehBodystyle were removed since each has only one value (SUV, Used)
- ListingID was removed due to logical irrelevance
- SellerCity, SellerState, SellerName were removed to keep Zipcode as the only seller information. Also having high unique values gives a high chance of overfitting.
- VehSellerNotes was removed due to its redundancy with VehFeats and other variables in the data
- VehColorInt was removed due to its high # of NA in both train and test data (over 10% of the data)

### Fill NA values in Test data

```
library(mice)
```

```
##
## Attaching package: 'mice'
```

```
## The following object is masked from 'package:stats':
##
## filter
```

```
## The following objects are masked from 'package:base':
##
## cbind, rbind
```

```
# Calculate mean of 'VehMileage'
mean_mileage <- mean(ucd_validation$VehMileage, na.rm = TRUE)
# Replace missing values with mean of column
ucd_validation$VehMileage <- ifelse(is.na(ucd_validation$VehMileage), mean_mileage, ucd_validation$VehMileage)
missing <- sapply(ucd_validation, function(x) sum(!nzchar(as.character(x))))
ucd_validation[ucd_validation == ""] <- NA
missing
```

```
## SellerIsPriv SellerListSrc SellerRating SellerRevCnt SellerZip
## 0 0 0 0 0
## VehCertified VehColorExt VehDriveTrain VehEngine VehFeats
## 0 7 64 58 37
## VehFuel VehHistory VehListdays VehMake VehMileage
## 0 27 0 0 0
## VehModel VehPriceLabel VehTransmission VehYear
## 0 38 27 0
```

This block of code is for imputation in R.(Filling NA values with the mode) Mode was chosen because for categorical values, there methods such as KNN or mean would not work.

```
missing2 <- sapply(ucd_validation, function(x) sum(!nzchar(as.character(x))))
missing2
```

```
## SellerIsPriv SellerListSrc SellerRating SellerRevCnt SellerZip
## 0 0 0 0 0
## VehCertified VehColorExt VehDriveTrain VehEngine VehFeats
## 0 0 0 0 0
## VehFuel VehHistory VehListdays VehMake VehMileage
## 0 0 0 0 0
## VehModel VehPriceLabel VehTransmission VehYear
## 0 0 0 0 0
```

## Removing abnormal values including NA

```
cat("ucd before removing empty values: ", dim(ucd), '\n')
```

```
## ucd before removing empty values: 6298 21
```

```
#removing na, or empty values
ucd <- na.omit(ucd)
ucd <- ucd[apply(ucd, 1, function(x) all(x != "")), ]
#result
cat("ucd after removing empty values: ", dim(ucd))
```

```
## ucd after removing empty values: 5441 21
```

After removing NA values, I removed cells that contained unnecessary special characters such as \$ # % @ &.

```
#Filter abnormal data with special characters like &, $, %, #, @, ` , ;
ucd <- ucd %>%
  filter(!(Vehicle_Trim %like% "&|%|\\$|#|@|`|;"))
```

## VehDriveTrain & VehTransmission column cleaning for both training and testing data

```
#VehDriveTrain change
#change Four wheel drive to 4WD #change Front wheel drive to FWD #change All wheel drive to AWD #change 4x4 to 4X
4
ucd$VehDriveTrain <- gsub("Four Wheel Drive", "4WD", ucd$VehDriveTrain)
ucd$VehDriveTrain <- gsub("Front Wheel Drive|FRONT-WHEEL DRIVE|Front-wheel Drive", "FWD", ucd$VehDriveTrain)
ucd$VehDriveTrain <- gsub("All Wheel Drive|All-wheel Drive|ALL WHEEL|AllWheelDrive|ALL-WHEEL DRIVE", "AWD", ucd$VehDriveTrain)
ucd$VehDriveTrain <- gsub("4x4", "4X4", ucd$VehDriveTrain)
#Remove rows
rows_to_filter_drivetrain <- c('4WD/AWD','ALL-WHEEL DRIVE WITH LOCKING AND LIMITED-SLIP DIFFERENTIAL','2WD')
#Rows select everything but the filter
ucd <- ucd[!(ucd$VehDriveTrain %in% rows_to_filter_drivetrain), ]

#Validation data
ucd_validation$VehDriveTrain <- gsub("Four Wheel Drive", "4WD", ucd_validation$VehDriveTrain)
ucd_validation$VehDriveTrain <- gsub("Front Wheel Drive|FRONT-WHEEL DRIVE|Front-wheel Drive", "FWD", ucd_validation$VehDriveTrain)
ucd_validation$VehDriveTrain <- gsub("All Wheel Drive|All-wheel Drive|ALL WHEEL|AllWheelDrive|ALL-WHEEL DRIVE", "AWD", ucd_validation$VehDriveTrain)
ucd_validation$VehDriveTrain <- gsub("4x4", "4X4", ucd_validation$VehDriveTrain)
```

```
#VehTransmission change
#change AUTOMATIC to Automatic #change Automatic 8-Speed & 8-Speed A/T & 8 Speed Automatic & Automatic, 8-Spd & 8
speed automatic & 8-SPEED AUTOMATIC to 8-Speed Automatic
ucd$VehTransmission <- gsub("AUTOMATIC|A|Automatic Transmission","Automatic", ucd$VehTransmission)
ucd$VehTransmission <- gsub("Automatic 8-Speed|8-Speed A/T|8 Speed Automatic|Automatic, 8-Spd|8 speed automatic|8
-SPEED AUTOMATIC|8-SPEED Automatic|8-SPEED A/T|8-speed Automatic|8 Spd Automatic","8-Speed Automatic", ucd$VehTransmission)
#Remove rows
count_T <- table(ucd$VehTransmission)
df_count_T <- count_T %>%
  as.data.frame() %>%
  arrange(desc(Freq))
freq1_T <- filter(df_count_T, Freq <= 2)
rows_to_filter_Transmission <- c(freq1_T$Var1)
# filter rows that contain any of the specified characters
ucd <- ucd[!(ucd$VehTransmission %in% rows_to_filter_Transmission), ]

# Validation data
ucd_validation$VehTransmission <- gsub("AUTOMATIC|A|Automatic Transmission","Automatic", ucd_validation$VehTransmission)
ucd_validation$VehTransmission <- gsub("Automatic 8-Speed|8-Speed A/T|8 Speed Automatic|Automatic, 8-Spd|8 speed automatic|8-SPEED AUTOMATIC|8-SPEED Automatic|8-SPEED A/T|8-speed Automatic|8 Spd Automatic","8-Speed Automatic", ucd_validation$VehTransmission)
```

```
skim(ucd)
```

#### Data summary

Name	ucd
Number of rows	5427
Number of columns	21
Column type frequency:	
character	12
logical	2

numeric

7

Group variables

None

**Variable type: character**

skim_variable	n_missing	complete_rate	min	max	empty	n_unique	whitespace
SellerListSrc	0	1	16	24	0	5	0
VehColorExt	0	1	2	41	0	160	0
VehDriveTrain	0	1	3	46	0	7	0
VehEngine	0	1	1	48	0	87	0
VehFeats	0	1	121	2004	0	741	0
VehFuel	0	1	6	13	0	4	0
VehHistory	0	1	7	110	0	32	0
VehMake	0	1	4	8	0	2	0
VehModel	0	1	3	14	0	2	0
VehPriceLabel	0	1	9	10	0	3	0
VehTransmission	0	1	7	35	0	15	0
Vehicle_Trim	0	1	3	32	0	27	0

**Variable type: logical**

skim_variable	n_missing	complete_rate	mean	count
SellerIsPriv	0	1	0.00	FAL: 5427
VehCertified	0	1	0.19	FAL: 4421, TRU: 1006

**Variable type: numeric**

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
SellerRating	0	1	4.18	1.12	0.00	4.0	4.60	4.80	5.00	
SellerRevCnt	0	1	449.54	1279.07	0.00	32.0	142.00	424.00	14635.00	
SellerZip	0	1	45333.80	20100.21	1105.00	29627.5	46804.00	60126.00	99654.00	
VehListdays	0	1	57.33	67.29	1.74	14.7	34.89	75.74	820.68	
VehMileage	0	1	26471.15	12978.81	5.00	17013.5	26181.00	36639.00	83037.00	
VehYear	0	1	2016.78	1.21	2015.00	2015.0	2017.00	2018.00	2019.00	
Dealer_Listing_Price	0	1	32132.36	7563.70	18289.00	26800.0	31000.00	35900.00	89500.00	

Among the list there are columns that we need to extract text values in a form that we can utilize. 1. Columns not to worry parsing(use factors to assign value): - "SellerListSrc" - "VehDriveTrain" - "VehFuel" - "VehMake" - "VehModel" - "VehPriceLabel" - "VehTransmission" - "Vehicle\_Trim"

## 2. Word

- "VehColorExt"
- "VehEngine"

Word refers to column values that should focus on specific words. For example, in VehColorExt column, color values could be better utilized when they are sorted out to specific colors that we desire to see. ex) Diamond Black Crystal Pearlcoat → 'Black' Collecting the key color from this description which is going to be one of the common colors 'Black' would be useful in training and predicting.

### 3. Phrase

- "VehFeats"
- "VehHistory"

Phrase refers to column values that should be parsed into phrases. For example, in VehFeats, it is crucial to separate the key features that the vehicle has. ex) ['4-Wheel Disc Brakes', 'ABS', 'Adjustable Steering Wheel', 'Aluminum Wheels', 'AM/FM Stereo', 'Automatic Headlights', 'Auxiliary Audio Input'] → 4-Wheel Disc Brakes, ABS, Adjustable Steering Wheel ... Collecting the key features would be crucial to the model training.

## Data Preprocessing

### Categorical Values Manual Preprocessing with Bag of Words & Concept of Multi-encoding

In order to utilize the word and phrase data as much as possible, sorting out certain words and phrases that appear often is going to be crucial. The following function parse\_list, parse\_row\_w\_space, parse\_row\_wo\_space, and encode\_row was written in order to parse the value and assign a value to encode categorical data into numbers.

#### Bag of words

Bag of words is a concept that gives count to a most appearing words in the text. Using the count, I have counted the most used features or histories of the used car. <https://www.codecademy.com/learn/dscp-natural-language-processing/modules/dscp-bag-of-words/cheatsheet> (<https://www.codecademy.com/learn/dscp-natural-language-processing/modules/dscp-bag-of-words/cheatsheet>) To make the sorting process with better accuracy, I am only going to use variables that has more than 100 counts. In order to assign values, I have normalized the values of the top 100 counts. (0~1)



```

parse_list <- function(x) {
  x <- gsub("^\\[|\\]$\"", "", x)
  elements <- strsplit(x, ", ")[[1]]
  elements <- gsub("'", "", elements) # remove apostrophes
  #elements <- gsub("*", "", elements) # remove apostrophes
  counts <- table(elements)
  data.frame(word = names(counts), count = as.numeric(counts), stringsAsFactors = FALSE)
}

parse_row_w_space <- function(row) {
  # Remove brackets and split by comma
  elements <- strsplit(gsub("^\\[|\\]$\"", "", row), ", ")[[1]]
  elements <- strsplit(gsub("^\\[|\\]$\"", "", row), " ")[[1]]
  # Remove any leading/trailing whitespace
  elements <- trimws(elements)
  elements <- gsub("'", "", elements) # remove apostrophes
  # Return vector of parsed elements
  elements_vec <- as.vector(elements)
  return(elements_vec)
}

parse_row_wo_space <- function(row) {
  # Remove brackets and split by comma
  elements <- strsplit(row, split = "..", fixed=TRUE)[[1]]
  elements <- strsplit(gsub("^\\[|\\]$\"", "", row), ", ")[[1]]
  #elements <- strsplit(gsub("^\\[/\\]$\"", "", row), " ")[[1]]
  # Remove any leading/trailing whitespace
  elements <- trimws(elements)
  elements <- gsub("'", "", elements) # remove apostrophes
  elements <- gsub("*", "", elements) # remove apostrophes
  # Return vector of parsed elements
  elements_vec <- as.vector(elements)
  return(elements_vec)
}

encode_row <- function(vector, weights_df) {
  # Parse the row to get the elements in the vector
  for (i in 1:length(vector)) {
    for (j in 1:length(vector[[i]])) {
      match_idx <- match(vector[[i]][j], weights_df$word)
      if (!is.na(match_idx)) {
        vector[[i]][j] <- weights_df$count[match_idx]
      }
      else {
        vector[[i]][j] <- 0.0
      }
    }
  }
  return(vector)
}

```

## Word separation & score

Use function `parse_row` & `encode_row` to separate the words or phrases and assign the counted value to the cell. Mechanism explained: 1. Vectorized the cell value - ex) ['ABS', 'Aluminum Wheels', 'AM/FM Stereo'] -> <ABS,Aluminum Wheels,AM/FM Stereo> 2. Counted each 100 frequent words or phrases - ex) ABS had 200 appearances in the data, Aluminum Wheels appeared 100 times, AM/FM Stereo appeared 50 times 3. Assign each value into the vector - ex) <ABS,Aluminum Wheels,AM/FM Stereo> -> <200,100,50> 4. Sum the vector values - ex) <200,100,50> -> 350

`word_score` function is meant for the training data, which contains the target value 'Vehicle Trim' and 'Price' `word_score_apply` is meant for the testing data.

```
#function format <- function(data, c("column_name1", "column_name2"), 'word'/'phrase')
```

```

#be sure to use after function 'parse_list', 'parse_row', and 'encode_row' is established
word_score <- function(df, columns, word_or_phrase, word_token) {
  for (i in columns) {
    #create additional dataframe for specific column
    print(i)
    temp_df <- df[i]
    #print(temp_df)
    #decide if you're going to prioritize a word or a phrase
    if (word_or_phrase == 'word') {
      print('word')
      #change the column name to "text"
      temp_df <- rename(temp_df, text = colnames(df[i]))
      #split text into individual words
      temp_df_words <- temp_df %>%
        unnest_tokens(word, text, to_lower = FALSE)
      #remove stop words (common words like "the", "a", etc.)
      temp_df_words <- temp_df_words %>%
        anti_join(stop_words)
      #count the frequency of each word
      word_freq <- temp_df_words %>%
        count(word, sort = TRUE)
      if (i %in% c('SellerCity', 'SellerName')) {
        top_words <- temp_df_words %>%
          count(word, sort = TRUE) %>%
          slice(1:2000)
      }
      else {
        #keep the top 100 most frequent words
        top_words <- temp_df_words %>%
          count(word, sort = TRUE) %>%
          slice(1:100)
      }

      #rename column 'n' as 'count'
      top_words <- rename(top_words, count = n)
      #print(top_words)
      #normalize the count of top words in a range of 0 to 1
      process <- preProcess(as.data.frame(top_words), method=c("range"))
      norm_temp_df <- predict(process, as.data.frame(top_words))
      word_token <- norm_temp_df
      #use function parse_row -> going to give vectors with split words ex) "hello world" -> <"hello", "world">
      temp_df_parsed <- t(apply(temp_df,1, parse_row_w_space))
    }
    #difference between word and phrase comes by the method of parsing the words
    else if (word_or_phrase == 'phrase') {
      print('phrase')
      temp_df_list <- temp_df %>%
        mutate(parsed = lapply(.,colnames(df[i])), parse_list)) %>%
        #split into each phrase
        unnest(parsed) %>%
        #create a dataframe that has phrase on the left and count of the phrase on the right
        group_by(word) %>%
        summarise(count = sum(count)) %>%
        #arrange in a descending order
        arrange(desc(count))
        #take only the top 100 phrases
        top_words_list <- temp_df_list[temp_df_list$count>100,]
        if (i %in% c('VehSellerNotes')) {
          top_words_list <- temp_df_list %>%
            slice(1:500)
        }
      else {
        #keep the top 100 most frequent words
        top_words_list <- temp_df_list %>%

```

```

        slice(1:100)
    }
    #normalize the count of top words in a range of 0 to 1
    top_words_list_normal <- copy(top_words_list)
    process <- preProcess(as.data.frame(top_words_list_normal), method=c("YeoJohnson","range"))
    norm_temp_df <- predict(process, as.data.frame(top_words_list_normal))
    #print(typeof(norm_temp_df))
    word_token <-< norm_temp_df
    #use function parse_row -> going to give vectors with split words ex) "hello world" -> <"hello", "world">
    temp_df_parsed <- t(apply(temp_df,1, parse_row_wo_space))
}
#use function encode_row -> if "hello" had frequency 30, and "world" had frequency 10 -> <'30','10'>
#but instead of 30 and 10, the normalized number(0 ~ 1) would go into the vectors
vector_number_temp_df <- t(encode_row(temp_df_parsed,norm_temp_df))
#convert into numeric form ex) <'30','10'> into <30,10> and sum the value
vector_row_sums <- apply(vector_number_temp_df, 1, function(x) sum(as.numeric(unlist(x))))
vector_row_sums_matrix <- matrix(vector_row_sums)
df[i] <- vector_row_sums_matrix
}
return(df)
}

```

These two methods were used to preprocess the data range: Normalize values so it ranges between 0 and 1 YeoJohnson: Like BoxCox, but works for negative values.

```

#function(data, c("column_name1","column_name2"), 'word'/'phrase')
#be sure to use after function 'parse_list', 'parse_row', and 'encode_row' is established
word_score_apply <- function(df, columns, word_or_phrase, cat_token) {
  for (i in columns) {
    #create additional dataframe for specific column
    print(i)
    temp_df <- df[i]
    #print(temp_df)
    #decide if you're going to prioritize a word or a phrase
    if (word_or_phrase == 'word') {
      print('word')
      #change the column name to "text"
      temp_df <- rename(temp_df, text = colnames(df[i]))
      #normalize the count of top words in a range of 0 to 1
      process <- preProcess(as.data.frame(cat_token), method=c("range"))
      norm_temp_df <- predict(process, as.data.frame(cat_token))
      #use function parse_row -> going to give vectors with split words ex) "hello world" -> <"hello", "world">
      temp_df_parsed <- t(apply(temp_df,1, parse_row_w_space))
    }
    #difference between word and phrase comes by the method of parsing the words
    else if (word_or_phrase == 'phrase') {
      print('phrase')
      #normalize the count of top words in a range of 0 to 1
      top_words_list_normal <- copy(cat_token)
      process <- preProcess(as.data.frame(top_words_list_normal), method=c("YeoJohnson","range"))
      norm_temp_df <- predict(process, as.data.frame(top_words_list_normal))
      #print(typeof(norm_temp_df))
      #use function parse_row -> going to give vectors with split words ex) "hello world" -> <"hello", "world">
      temp_df_parsed <- t(apply(temp_df,1, parse_row_wo_space))
    }
    #use function encode_row -> if "hello" had frequency 30, and "world" had frequency 10 -> <'30','10'>
    #but instead of 30 and 10, the normalized number(0 ~ 1) would go into the vectors
    vector_number_temp_df <- t(encode_row(temp_df_parsed,norm_temp_df))
    #convert into numeric form ex) <'30','10'> into <30,10> and sum the value
    vector_row_sums <- apply(vector_number_temp_df, 1, function(x) sum(as.numeric(unlist(x))))
    vector_row_sums_matrix <- matrix(vector_row_sums)
    df[i] <- vector_row_sums_matrix
  }
  return(df)
}

```

After the function was formed, the training data was filtered.

Rest of the columns that has categorical values were transformed into factors.

```

ucd <- ucd %>% mutate_at(vars(SellerListSrc, VehDriveTrain, VehFuel, VehMake, VehModel, VehPriceLabel, VehTransmi
ssion,Vehicle_Trim), factor)

```

After all the preprocess, I double checked if there were any na values.

```

nrows_with_na <- sum(is.na(ucd))
cat("Number of rows with NA values:", nrows_with_na, "\n")

```

```
## Number of rows with NA values: 0
```

## Training and Testing

### Training data

Assign target variable

```
ucd_train_price <- select(ucd, -Vehicle_Trim)
ucd_train_Trim <- select(ucd, -Dealer_Listing_Price)
```

I used 'caret' package for training. Control is for cross validation method, which has 5 folds. Metric was assigned to base on Area under the curve.

```
control <- trainControl(method="cv", number=5)
metric <- "ROC AUC"
```

These are the list of models used for training Price 1. lm 2. glm 3. sgd 4. knn 5. svm 6. cart 7. rf 8. et

Trim 1. knn 2. svm 3. cart 4. rf 5. et

```
## Trim

set.seed(100)

# kNN
Trim.knn <- train(Vehicle_Trim~., data=ucd_train_Trim, method="knn", metric=metric, trControl=control)
# SVM
Trim.svm <- train(Vehicle_Trim~., data=ucd_train_Trim, method="svmRadial", metric=metric, trControl=control)
# Classification and regression trees
Trim.cart <- train(Vehicle_Trim~., data=ucd_train_Trim, method="rpart", metric=metric, trControl=control)
# Random Forest
Trim.rf <- train(Vehicle_Trim~., data=ucd_train_Trim, method="rf", tuneLength=5, metric=metric, trControl=control)
# ExtraTree
Trim.et <- train(Vehicle_Trim~., data=ucd_train_Trim, method="ranger", metric=metric, trControl=control)

pricerresults <- resamples(list(lm=Price.lm,glm=Price.glm,sgd=Price.sgd,knn=Price.knn,svm=Price.svm,car
t,rf=Price.rf,et=Price.et))
trimresults <- resamples(list(knn=Trim.knn,svm=Trim.svm,car=Trim.cart,rf=Trim.rf,et=Trim.et))
summary(pricerresults)
```

```
##
## Call:
## summary.resamples(object = pricerresults)
##
## Models: lm, glm, sgd, knn, svm, cart, rf, et
## Number of resamples: 5
##
## MAE
##      Min.   1st Qu.   Median     Mean   3rd Qu.     Max. NA's
## lm    2824.009 2882.507 2900.598 2904.272 2918.202 2996.043    0
## glm    2761.335 2836.442 2917.534 2907.580 3003.682 3018.908    0
## sgd    2829.606 2842.239 2860.615 2886.934 2902.388 2999.825    0
## knn    4043.781 4060.873 4163.030 4183.693 4227.138 4423.643    0
## svm    5373.250 5382.382 5390.172 5394.554 5404.700 5422.268    0
## cart   3890.104 3995.837 4013.871 4061.666 4191.227 4217.292    0
## rf     1871.959 1875.993 1927.400 1917.065 1931.686 1978.285    0
## et     1837.852 1897.545 1919.068 1918.291 1944.862 1992.127    0
##
## RMSE
##      Min.   1st Qu.   Median     Mean   3rd Qu.     Max. NA's
## lm    3820.021 4222.287 4581.713 4398.678 4675.060 4694.309    0
## glm    4028.511 4284.161 4328.152 4412.415 4582.628 4838.624    0
## sgd    3931.580 4234.240 4339.176 4414.579 4582.402 4985.499    0
## knn    5863.997 5883.404 6081.272 6141.382 6238.188 6640.047    0
## svm    7139.311 7445.400 7516.271 7500.810 7541.995 7861.071    0
## cart   5699.026 5849.488 6069.694 6105.917 6438.108 6473.269    0
## rf     2940.435 2967.333 2991.189 3047.419 3039.435 3298.701    0
## et     2718.865 3014.730 3064.684 3023.603 3130.794 3188.940    0
##
## Rsquared
##      Min.   1st Qu.   Median     Mean   3rd Qu.     Max. NA's
## lm    0.6251944 0.6509154 0.6723344 0.6642649 0.6852628 0.6876177    0
## glm    0.6034958 0.6429867 0.6821870 0.6611349 0.6840934 0.6929117    0
## sgd    0.6333568 0.6493347 0.6543920 0.6617603 0.6655810 0.7061367    0
## knn    0.3254704 0.3271286 0.3329277 0.3515018 0.3738125 0.3981698    0
## svm    0.2469491 0.2660060 0.2886244 0.2775477 0.2892987 0.2968603    0
## cart   0.3207859 0.3440293 0.3450173 0.3508130 0.3649653 0.3792674    0
## rf     0.8221699 0.8261245 0.8490976 0.8416622 0.8539127 0.8570063    0
## et     0.8257190 0.8308445 0.8324768 0.8436477 0.8614027 0.8677956    0
```

```
summary(trimresults)
```

```
##
## Call:
## summary.resamples(object = trimresults)
##
## Models: knn, svm, cart, rf, et
## Number of resamples: 5
##
## Accuracy
##           Min.    1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## knn  0.362790698 0.3695924 0.3763941 0.3718596 0.3763941 0.3763941    2
## svm  0.007462687 0.3163265 0.4016775 0.3076925 0.4035250 0.4094708    0
## cart 0.640000000 0.6437209 0.6514019 0.6517141 0.6598703 0.6635774    0
## rf   0.825278810 0.8267384 0.8281979 0.8297105 0.8319263 0.8356546    2
## et   0.818435754 0.8192771 0.8212291 0.8285618 0.8409302 0.8429368    0
##
## Kappa
##           Min.    1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## knn  0.173686567 0.17991074 0.1861349 0.1824723 0.1868652 0.1875956    2
## svm  0.001596594 0.08479266 0.1613207 0.1205500 0.1757028 0.1793371    0
## cart 0.533688426 0.53803076 0.5491286 0.5516087 0.5672397 0.5699562    0
## rf   0.788616374 0.79009357 0.7915708 0.7937152 0.7962646 0.8009585    2
## et   0.780517481 0.78152792 0.7834802 0.7925723 0.8077595 0.8095763    0
```

## Training RMSE, R2 for Price model and Accuracy and Kappa for Trim model

```
price_rmse_values <- data.frame(
  Price_Model = c("lm", "glm", "sgd", "knn", "svm", "cart", "rf", "et"),
  Price_RMSE = c(pricerresults$values$`lm~RMSE`, pricerresults$values$`glm~RMSE`, pricerresults$values$`sgd~RMSE`, pricerresults$values$`knn~RMSE`, pricerresults$values$`svm~RMSE`, pricerresults$values$`cart~RMSE`, pricerresults$values$`rf~RMSE`, pricerresults$values$`et~RMSE`),
  Price_R2 = c(pricerresults$values$`lm~Rsquared`, pricerresults$values$`glm~Rsquared`, pricerresults$values$`sgd~Rsquared`, pricerresults$values$`knn~Rsquared`, pricerresults$values$`svm~Rsquared`, pricerresults$values$`cart~Rsquared`, pricerresults$values$`rf~Rsquared`, pricerresults$values$`et~Rsquared`)
)

trim_rmse_values <- data.frame(
  Trim_Model = c("knn", "svm", "cart", "rf", "et"),
  Trim_Accuracy = c(trimresults$values$`knn~Accuracy`, trimresults$values$`svm~Accuracy`, trimresults$values$`cart~Accuracy`, trimresults$values$`rf~Accuracy`, trimresults$values$`et~Accuracy`),
  Trim_Kappa = c(trimresults$values$`knn~Kappa`, trimresults$values$`svm~Kappa`, trimresults$values$`cart~Kappa`, trimresults$values$`rf~Kappa`, trimresults$values$`et~Kappa`)
)
```

## Testing data

```
ucd_test2 <- ucd_test
ucd_test3 <- ucd_test2 # Duplicate test data set
# Create a vector of factor column names in ucd_test3
factor_cols <- names(ucd_test3)[sapply(ucd_test3, is.factor)]
# Loop over factor columns and replace values not in ucd with NA

for (col in factor_cols) {
  ucd_test3[[col]][!(ucd_test3[[col]] %in% unique(ucd[[col]]))] <- NA
}
```

```
missing2 <- sapply(ucd_test3, function(x) sum(!nzchar(as.character(x))))
ucd_test3[ucd_test3 == ""] <- NA
```

## Prediction

```
head(price_predictions)
```

```
## predictions...1 predictions...2 predictions...3 predictions...4
## 1 37554.57 37554.57 37588.70 35036.44
## 2 28119.25 28119.25 28121.79 27012.78
## 3 20850.45 20850.45 21060.34 24800.22
## 4 23912.57 23912.57 23992.92 26081.89
## 5 37970.74 37970.74 37963.02 36939.78
## 6 32604.66 32604.66 32424.77 31563.89
## predictions...5 predictions...6 predictions...7 predictions...8
## 1 31430.28 37561.59 36863.38 37152.82
## 2 30609.09 27850.39 26229.47 26378.20
## 3 30916.40 27850.39 25484.54 25588.82
## 4 30651.94 27850.39 24774.53 24766.67
## 5 31173.47 37561.59 37804.42 37586.89
## 6 31314.17 27850.39 29962.62 29945.57
```

```
head(trim_predictions)
```

```
## predictions...1 predictions...2 predictions...3 predictions...4
## 1 Premium Luxury Premium Luxury Premium Luxury Premium Luxury
## 2 Limited Limited Limited Limited
## 3 Limited Limited Overland Overland
## 4 Limited Limited Limited Limited
## 5 Premium Luxury Premium Luxury Luxury Base
## 6 Limited Limited Limited Limited
## predictions...5
## 1 Premium Luxury
## 2 Limited
## 3 Overland
## 4 Limited
## 5 Base
## 6 Limited
```

## Choosing model

ExtraTree method was showed good R2 and RMSE value for Price model.

Random Forest method showed good Accuracy and Kappa value for Trim model

Price:

1. lm 2. glm 3. sgd 4. knn 5. svm 6. cart 7. rf 8. et

Trim:

1. knn 2. svm 3. cart 4. rf 5. et

```
price_final <- price_predictions$predictions...8
trim_final <- trim_predictions$predictions...4
```

```
price_final <- as.data.frame(price_final)
trim_final <- as.data.frame(trim_final)
```

```
dim(trim_final)
```

```
## [1] 1000 1
```

```
result_submit <- cbind(list_ID,trim_final,price_final)
```

```
write.csv(result_submit, "submit_file.csv", row.names = FALSE)
```



Appendix: Bag of words: used to preprocess the data to give weights to the categorical values based on the appearances in the data.

<https://www.codecademy.com/learn/dscp-natural-language-processing/modules/dscp-bag-of-words/cheatsheet>

(<https://www.codecademy.com/learn/dscp-natural-language-processing/modules/dscp-bag-of-words/cheatsheet>)