IDSHW4.R

Vaibhav Khurana

2023-03-31

```
#Clear the environment
rm(list=ls())

#Libraries
library(dplyr)
library(ggplot2)

#PART-1
#List the files available in the working directory
list.files()
```

```
## [1] "IDS472HW4.Rproj"      "IDS472HW4diamond.csv" "IDSHW4.R"             "IDSHW4.spin.R"
"IDSHW4.spin.Rmd"
```

```
#(1A) Load the converted file (excel to csv) and analyse the structure of the variables present.
diamonds <- read.csv("IDS472HW4diamond.csv")

#check if data is not misisng
test_row <- diamonds[7000,]
print(test_row)
```

```
##        ID Carat.Weight   Cut Color Clarity Polish Symmetry Report Price
## 7000 7000          2.5 Ideal     H     VS1     VG       EX    GIA
```

```
#delete the rows with missing Price value for appropriate data preparation
diamonds <- diamonds[!diamonds$Price == "",]
str(diamonds)
```

```
## 'data.frame':    6000 obs. of  9 variables:
##  $ ID          : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Carat.Weight: num  1.1 0.83 0.85 0.91 0.83 1.53 1 1.5 2.11 1.05 ...
##  $ Cut         : chr  "Ideal" "Ideal" "Ideal" "Ideal" ...
##  $ Color       : chr  "H" "H" "H" "E" ...
##  $ Clarity     : chr  "SI1" "VS1" "SI1" "SI1" ...
##  $ Polish      : chr  "VG" "ID" "EX" "VG" ...
##  $ Symmetry    : chr  "EX" "ID" "EX" "VG" ...
##  $ Report      : chr  "GIA" "AGSL" "GIA" "GIA" ...
##  $ Price       : chr  "$5,169 " "$3,470 " "$3,183 " "$4,370 " ...
```

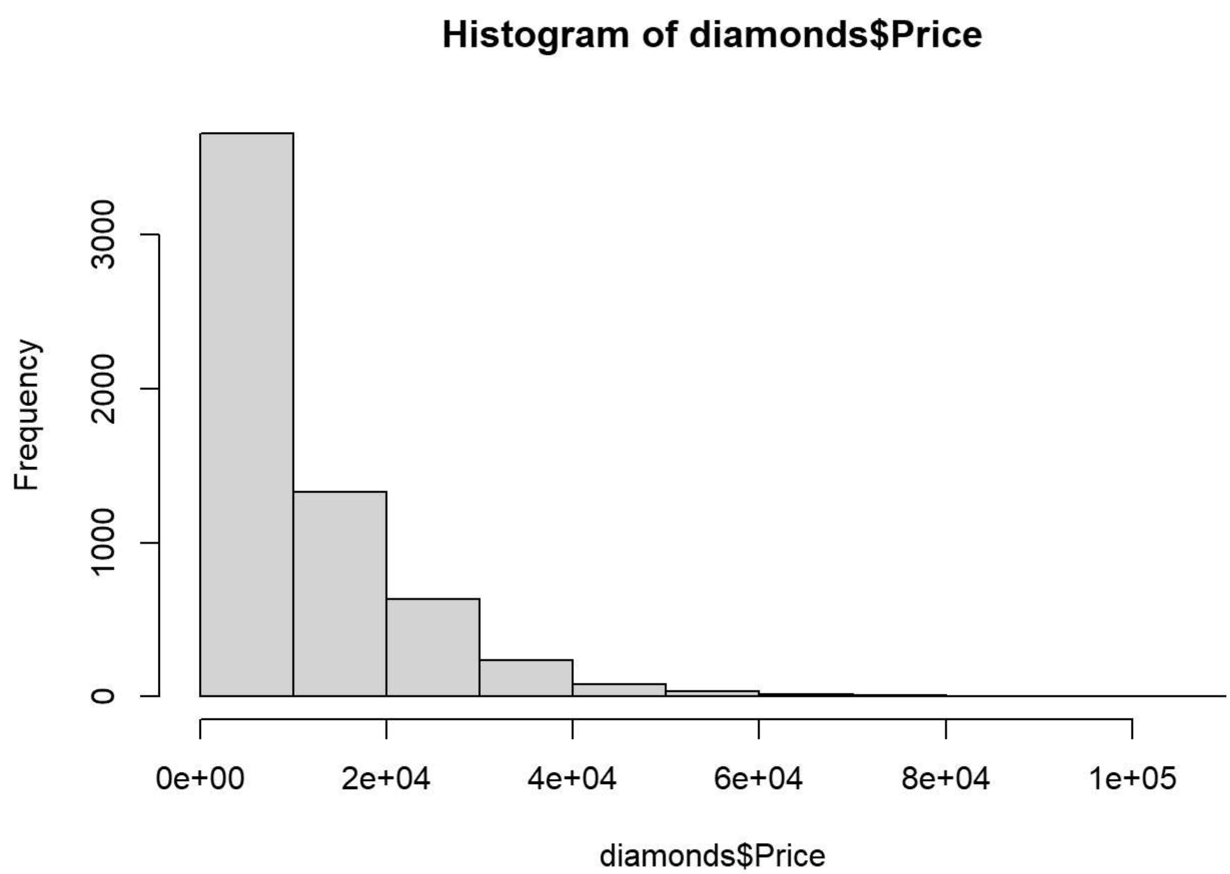
```
#(1B) Exploratory data analysis.
# remove the dollar sign from the Price column.
diamonds$Price <- gsub("[\\$,]", "", diamonds$Price)
# Convert from character to numeric.
diamonds$Price <- as.numeric(diamonds$Price)


#find outliers in price
summary(diamonds$Price)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    2184    5150    7857   11792   15036  101561
```

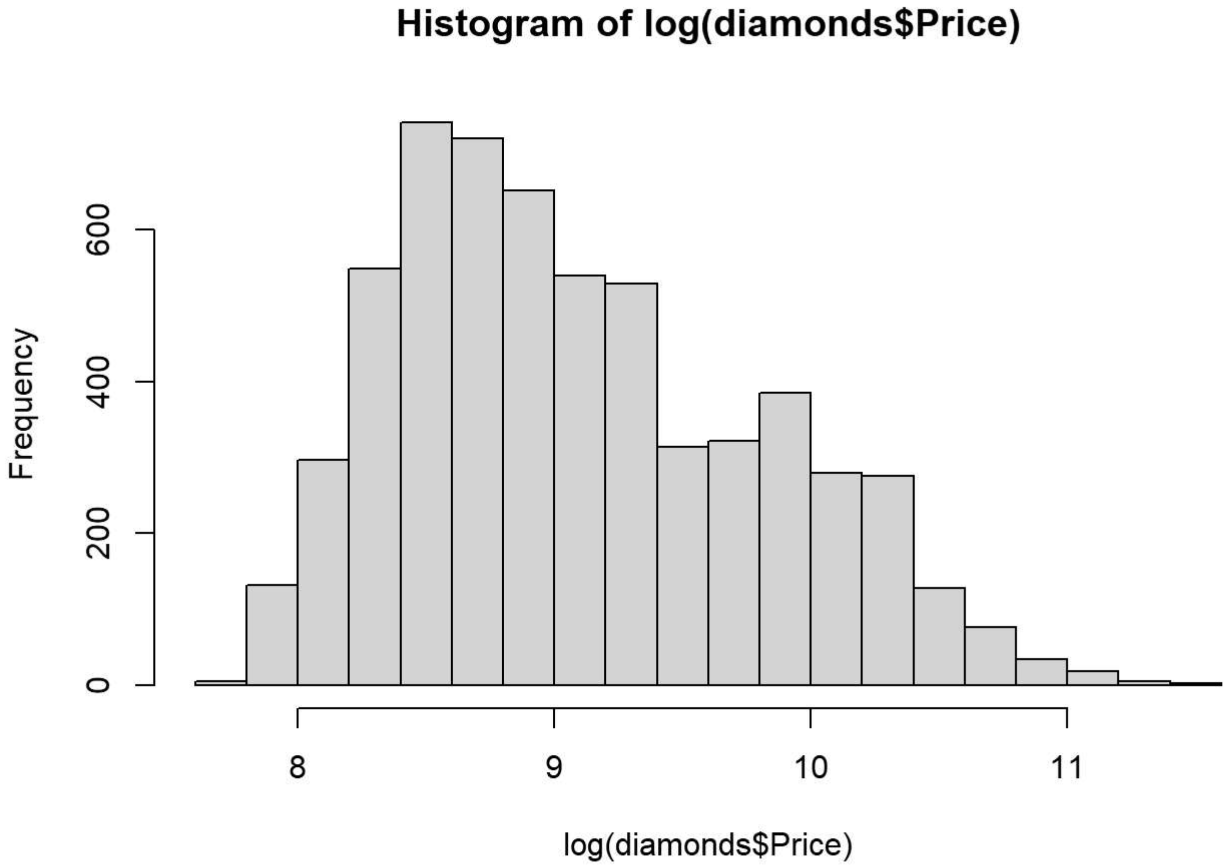
```
sd(diamonds$Price)
```

```
## [1] 10184.35
```

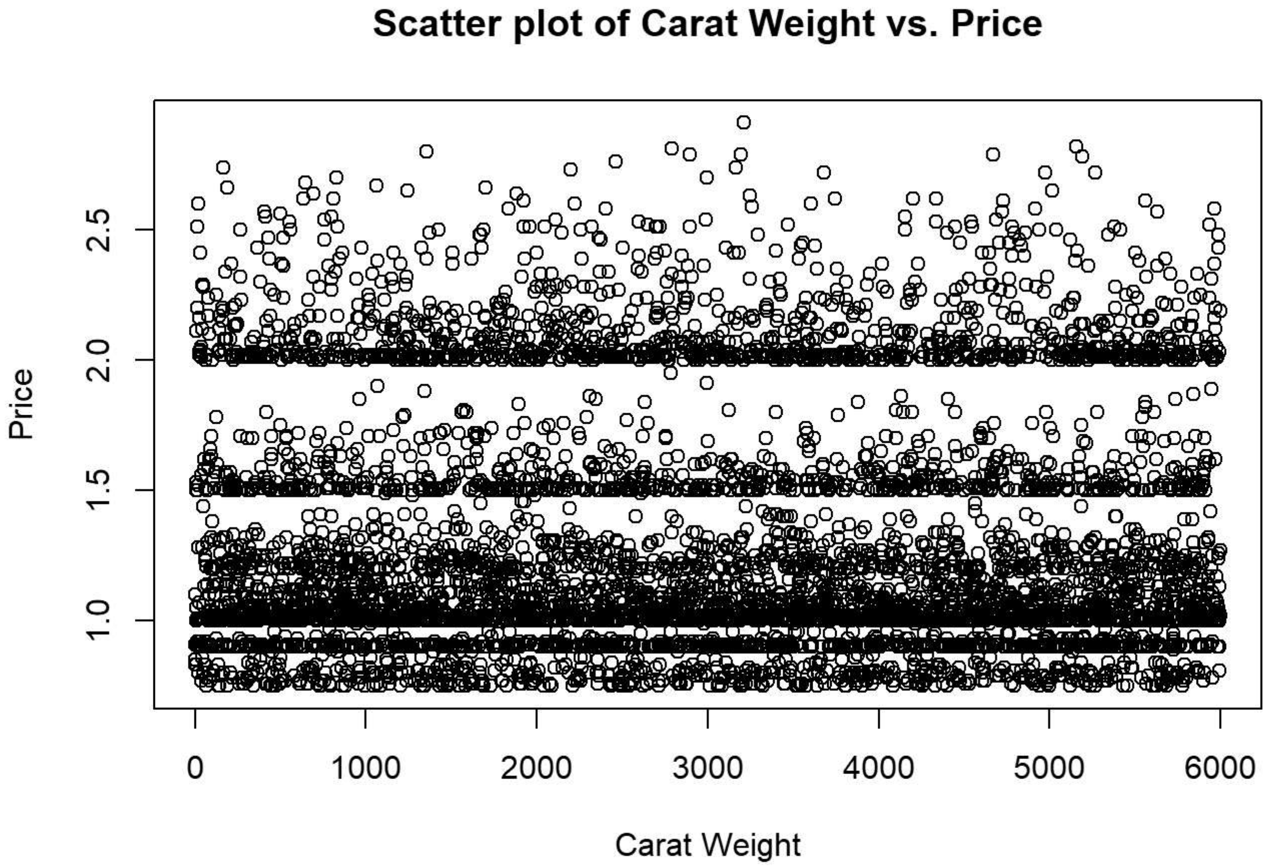
```
hist(diamonds$Price)
```



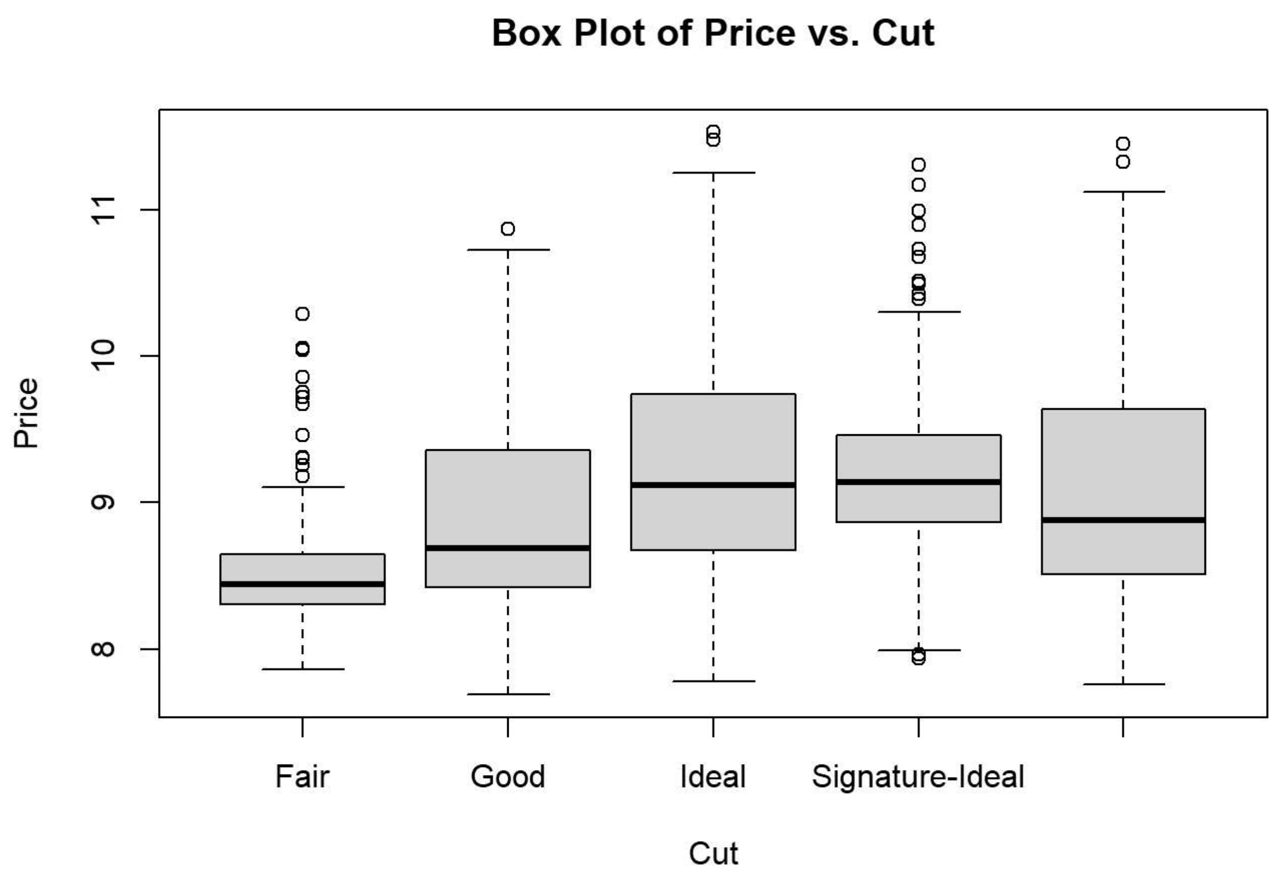

```
hist(log(diamonds$Price))
```



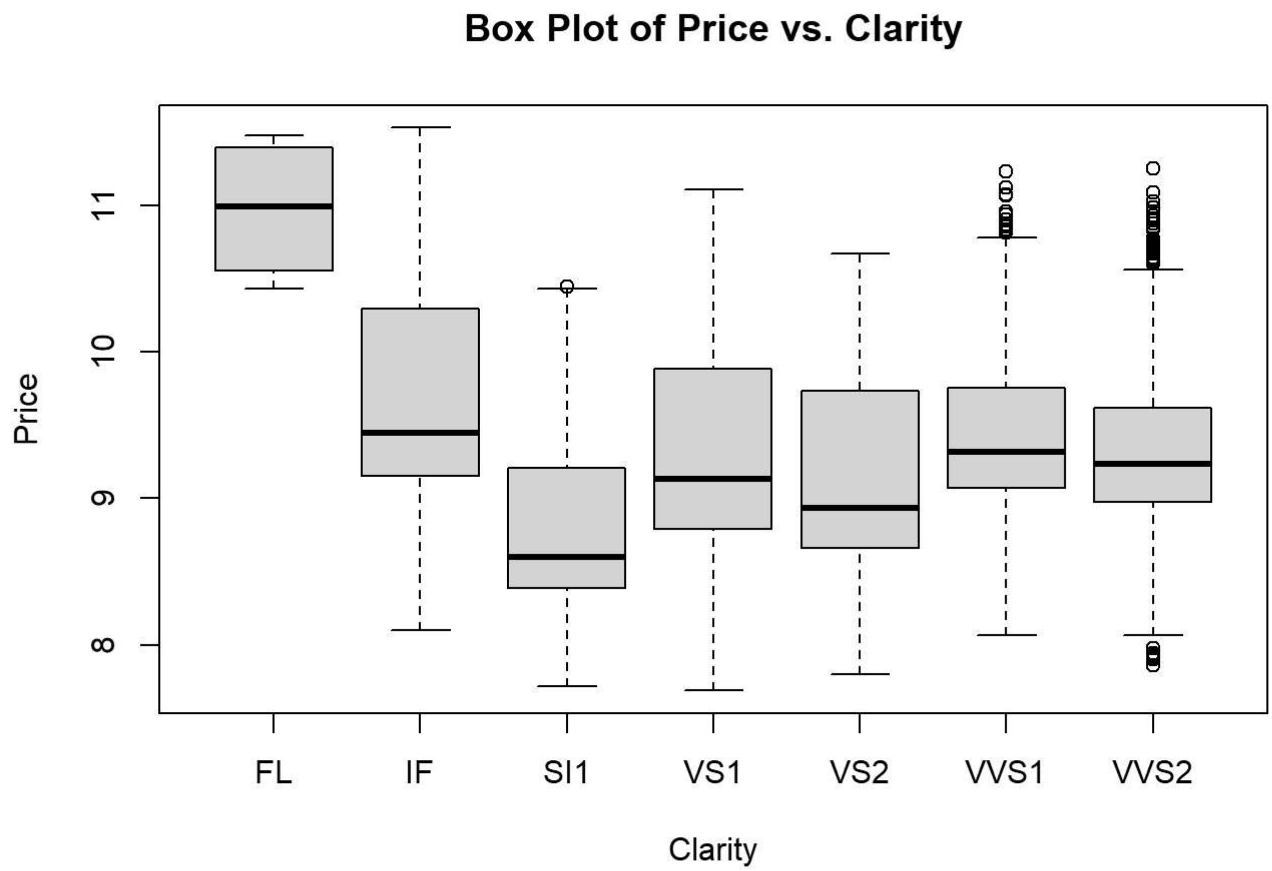

```
#handle outliers
diamonds$Price <- log(diamonds$Price)
View(diamonds)

# create a scatter plot of carat weight against price
plot(diamonds$Carat.Weight, diamonds$price, main="Scatter plot of Carat Weight vs. Price", xlab
="Carat Weight", ylab="Price")
```




```
# create a box plot of price against cut
boxplot(Price ~ Cut, data=diamonds, main="Box Plot of Price vs. Cut", xlab="Cut", ylab="Price")
```




```
# create a box plot of price against clarity
boxplot(Price ~ Clarity, data=diamonds, main="Box Plot of Price vs. Clarity", xlab="Clarity", yl
ab="Price")
```




```
#(1C) Splitiing the data set into 70% and %30
#Convert all characters into factors for easier analysis
diamonds[sapply(diamonds, is.character)] <- lapply(diamonds[sapply(diamonds, is.character)], as.
factor)
summary(diamonds)
```

```
##        ID        Carat.Weight                Cut       Color    Clarity     Polish    Symmetr
y   Report         Price
##  Min.   :   1   Min.   :0.750   Fair           : 129   D: 661   FL  :   4   EX:2425   EX:2059
AGSL: 734   Min.   : 7.689
##  1st Qu.:1501   1st Qu.:1.000   Good           : 708   E: 778   IF  : 219   G : 571   G : 916
GIA :5266   1st Qu.: 8.547
##  Median :3000   Median :1.130   Ideal          :2482   F:1013   SI1 :2059   ID: 595   ID: 608
Median : 8.969
##  Mean   :3000   Mean   :1.335   Signature-Ideal: 253   G:1501   VS1 :1192   VG:2409   VG:2417
Mean   : 9.100
##  3rd Qu.:4500   3rd Qu.:1.590   Very Good      :2428   H:1079   VS2 :1575
3rd Qu.: 9.618
##  Max.   :6000   Max.   :2.910                          I: 968   VVS1: 285
Max.   :11.528
##                                                                 VVS2: 666
```

```r
# split the dataset
set.seed(123)
train_index <- sample(2, 6000, prob = c(0.7, 0.3), replace = T)

train <- diamonds[train_index == 1, ]
test <- diamonds[train_index == 2, ]

#(1D) Train a linear regression model
linreg_model <- lm(formula = Price ~ .,
                   data = train)

#(1D) AND (1E)
# Check number of significant variables and R^2
summary(linreg_model)
```

```
##
## Call:
## lm(formula = Price ~ ., data = train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.69044 -0.06931  0.02401  0.09171  0.56798
##
## Coefficients:
##                      Estimate Std. Error t value Pr(>|t|)
## (Intercept)         8.134e+00  8.496e-02  95.731  < 2e-16 ***
## ID                  6.433e-07  1.256e-06   0.512 0.608568
## Carat.Weight        1.364e+00  4.698e-03 290.257  < 2e-16 ***
## CutGood             5.635e-02  1.668e-02   3.378 0.000736 ***
## CutIdeal            1.125e-01  1.630e-02   6.901 5.95e-12 ***
## CutSignature-Ideal  2.681e-01  1.994e-02  13.448  < 2e-16 ***
## CutVery Good        8.157e-02  1.590e-02   5.129 3.05e-07 ***
## ColorE             -9.513e-02  8.987e-03 -10.585  < 2e-16 ***
## ColorF             -1.120e-01  8.441e-03 -13.273  < 2e-16 ***
## ColorG             -1.927e-01  8.006e-03 -24.073  < 2e-16 ***
## ColorH             -3.284e-01  8.433e-03 -38.937  < 2e-16 ***
## ColorI             -4.654e-01  8.672e-03 -53.660  < 2e-16 ***
## ClarityIF          -4.023e-01  8.289e-02  -4.853 1.26e-06 ***
## ClaritySI1         -9.812e-01  8.229e-02 -11.923  < 2e-16 ***
## ClarityVS1         -7.300e-01  8.233e-02  -8.866  < 2e-16 ***
## ClarityVS2         -8.201e-01  8.233e-02  -9.962  < 2e-16 ***
## ClarityVVS1        -5.257e-01  8.284e-02  -6.345 2.46e-10 ***
## ClarityVVS2        -5.857e-01  8.246e-02  -7.103 1.43e-12 ***
## PolishG            -3.716e-02  9.088e-03  -4.089 4.40e-05 ***
## PolishID            7.538e-02  3.250e-02   2.319 0.020441 *
## PolishVG           -2.166e-02  5.663e-03  -3.825 0.000133 ***
## SymmetryG          -1.879e-02  8.521e-03  -2.205 0.027492 *
## SymmetryID          5.379e-03  3.368e-02   0.160 0.873139
## SymmetryVG         -1.233e-02  5.969e-03  -2.066 0.038847 *
## ReportGIA           8.423e-02  1.509e-02   5.583 2.51e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1412 on 4186 degrees of freedom
## Multiple R-squared:  0.961,  Adjusted R-squared:  0.9607
## F-statistic:  4294 on 24 and 4186 DF,  p-value: < 2.2e-16
```

```
'Number of variables significant in the model is 19'
```

```
## [1] "Number of variables significant in the model is 19"
```

```
'R^2 value of the model is 96%'
```

```
## [1] "R^2 value of the model is 96%"
```

```
#(1F) MAPE
# MAPE on training data
linreg_trainpreds <- (linreg_model$fitted.values)
err_train <- exp(linreg_trainpreds) - exp(train$Price)
abserr_train <- abs(err_train)
percabserr_train <- abserr_train / exp(train$Price)
mape_train <- mean(percabserr_train)
mape_train
```

```
## [1] 0.1099071
```

```
#(1G) Over priced diamond according to the model

over_priced <- train %>% mutate(pred_error =exp(linreg_trainpreds) - exp(train$Price)) %>% filte
r(pred_error == max(pred_error))
print(over_priced)
```

```
##     ID Carat.Weight   Cut Color Clarity Polish Symmetry Report    Price pred_error
## 1 5156         2.82 Ideal     H     VS1     ID       ID   AGSL 10.42668   33565.48
```

```
#(1H) MAPE on testing data
linreg_testpreds <- predict(linreg_model, test)
err_test <- exp(linreg_testpreds) - exp(test$Price)
abserr_test <- abs(err_test)
percabserr_test <- abserr_test / exp(test$Price)
mape_test <- mean(percabserr_test)
mape_test
```

```
## [1] 0.1124046
```

```
#(1I) # assuming the training dataset is called "diamonds_train"
recommended_diamond <- train %>% mutate(predicted_price = (linreg_model$fitted.values)) %>%
  filter(predicted_price <= log(12000)) %>%
  slice_max(Carat.Weight)

# output the details of the recommended diamond
print(recommended_diamond)
```

```
##     ID Carat.Weight       Cut Color Clarity Polish Symmetry Report    Price predicted_price
## 1 1589          1.8 Very Good     I     SI1     VG        G    GIA 9.375261        9.267911
```

```
#PART-2
#preparing the data for model building
library(fastDummies)
diamonds2 <- fastDummies::dummy_cols(diamonds)
diamonds2 <- diamonds2 %>% select(-Cut, -Color, -Clarity, -Polish, -Symmetry, -Report, -ID)

# You need to scale the data
mins <- apply(diamonds2, 2, min)
maxs <- apply(diamonds2, 2, max)
diamonds3 <- scale(diamonds2, mins, maxs-mins) # (x[j,i] - min[,i]) / (max[,i]-min[,i])

train_nn <- diamonds3[train_index == 1, ]
test_nn <- diamonds3[train_index == 2, ]

#Convert arrays to datasets
train_nn <- as.data.frame(train_nn)
test_nn <- as.data.frame(test_nn)

library(nnet)
#(2A) Building neaural network
# Define a function to train and test a neural network model
train_and_test <- function(num_neurons) {
  # Train the model
  nn_model <- nnet(Price ~ ., data = train_nn, size = num_neurons, linout = F, decay=0.01, maxit
=100)

  # Make predictions on the training dataset
  train_preds <- predict(nn_model, train_nn)
  train_preds <- train_preds * (maxs[2] - mins[2]) + mins[2]
  train_err <- exp(train_preds) - exp(train$Price)
  train_abserr <- abs(train_err)
  train_percabserr <- train_abserr / exp(train$Price)
  train_mape <- mean(train_percabserr)
  #train_mape <- mean(abs(exp(train_nn$Price) - exp(train_preds)) / exp(train_nn$Price))

  # Make predictions on the testing dataset
  test_preds <- predict(nn_model, test_nn)
  test_preds <- test_preds * (maxs[2] - mins[2]) + mins[2]
  test_err <- exp(test_preds) - exp(test$Price)
  test_abserr <- abs(test_err)
  test_percabserr <- test_abserr / exp(test$Price)
  test_mape <- mean(test_percabserr)
  #test_mape <- mean(abs(exp(test_nn$Price) - exp(test_preds)) / exp(test_nn$Price))

  # Return the results
  c(num_neurons, train_mape, test_mape)
}

# Specify the number of neurons to test
num_neurons_list <- c(1, 5, 10, 20)
```

```r
# Train and test the models with varying number of neurons
results <- t(sapply(num_neurons_list, train_and_test))
```

```
## # weights:  32
## initial  value 422.804250
## iter  10 value 27.786606
## iter  20 value 10.636107
## iter  30 value 7.683301
## iter  40 value 6.475460
## iter  50 value 6.281204
## iter  60 value 6.247146
## iter  70 value 6.222661
## iter  80 value 6.196591
## iter  90 value 6.185530
## iter 100 value 6.173111
## final  value 6.173111
## stopped after 100 iterations
## # weights:  156
## initial  value 173.638901
## iter  10 value 48.832678
## iter  20 value 9.900606
## iter  30 value 5.897177
## iter  40 value 4.603062
## iter  50 value 3.773618
## iter  60 value 3.552638
## iter  70 value 3.452973
## iter  80 value 3.365625
## iter  90 value 3.290889
## iter 100 value 3.226602
## final  value 3.226602
## stopped after 100 iterations
## # weights:  311
## initial  value 383.283343
## iter  10 value 22.707607
## iter  20 value 6.731383
## iter  30 value 4.405760
## iter  40 value 3.797824
## iter  50 value 3.446789
## iter  60 value 3.313134
## iter  70 value 3.246431
## iter  80 value 3.171890
## iter  90 value 3.105898
## iter 100 value 3.060064
## final  value 3.060064
## stopped after 100 iterations
## # weights:  621
## initial  value 343.421962
## iter  10 value 37.330589
## iter  20 value 7.310553
## iter  30 value 5.695979
## iter  40 value 4.611188
## iter  50 value 3.825766
## iter  60 value 3.458326
## iter  70 value 3.320896
## iter  80 value 3.219583
```

```
## iter  90 value 3.144880
## iter 100 value 3.100968
## final  value 3.100968
## stopped after 100 iterations
```

```
# Add column names to the results table
colnames(results) <- c("num_neurons", "train_mape", "test_mape")

# Print the results table
print(results)
```

```
##      num_neurons train_mape  test_mape
## [1,]           1 0.10783511 0.11154691
## [2,]           5 0.06801746 0.07117622
## [3,]          10 0.06525302 0.06840965
## [4,]          20 0.06578899 0.06882397
```

```
#Now run the optimal neural network and print its summary
nn_Omodel <- nnet(Price ~ ., data = train_nn, size = 10, linout = F, decay=0.01, maxit=100)
```

```
## # weights:  311
## initial  value 268.719993
## iter  10 value 13.003925
## iter  20 value 5.586488
## iter  30 value 4.583167
## iter  40 value 3.978168
## iter  50 value 3.552750
## iter  60 value 3.355300
## iter  70 value 3.241438
## iter  80 value 3.195175
## iter  90 value 3.160229
## iter 100 value 3.131532
## final  value 3.131532
## stopped after 100 iterations
```

```
summary(nn_Omodel)
```

```
## a 29-10-1 network with 311 weights
## options were - decay=0.01
##   b->h1  i1->h1  i2->h1  i3->h1  i4->h1  i5->h1  i6->h1  i7->h1  i8->h1  i9->h1 i10->h1 i11->
h1 i12->h1 i13->h1 i14->h1
##    0.63    0.12    0.07    0.10    0.40   -0.03    0.09    0.19   -0.03    0.37    0.44   -0.
12   -0.21    0.02    0.20
## i15->h1 i16->h1 i17->h1 i18->h1 i19->h1 i20->h1 i21->h1 i22->h1 i23->h1 i24->h1 i25->h1 i26->
h1 i27->h1 i28->h1 i29->h1
##   -0.13    0.28    0.13    0.06    0.11    0.36    0.17    0.18   -0.07    0.34    0.07    0.
20    0.06    0.14    0.51
##   b->h2  i1->h2  i2->h2  i3->h2  i4->h2  i5->h2  i6->h2  i7->h2  i8->h2  i9->h2 i10->h2 i11->
h2 i12->h2 i13->h2 i14->h2
##   -0.85    1.70   -0.03   -0.26   -0.25   -0.05   -0.29    0.66    0.11    0.04   -0.63   -0.
44   -0.61    0.27    0.70
## i15->h2 i16->h2 i17->h2 i18->h2 i19->h2 i20->h2 i21->h2 i22->h2 i23->h2 i24->h2 i25->h2 i26->
h2 i27->h2 i28->h2 i29->h2
##   -0.63   -0.34   -0.81    0.20   -0.26   -0.19   -0.25   -0.29   -0.14   -0.15   -0.26   -0.
31   -0.13   -0.37   -0.49
##   b->h3  i1->h3  i2->h3  i3->h3  i4->h3  i5->h3  i6->h3  i7->h3  i8->h3  i9->h3 i10->h3 i11->
h3 i12->h3 i13->h3 i14->h3
##    0.20   -0.40    0.15    0.08    0.09   -0.19    0.07   -0.09   -0.02    0.28    0.02    0.
01    0.01   -0.07    0.02
## i15->h3 i16->h3 i17->h3 i18->h3 i19->h3 i20->h3 i21->h3 i22->h3 i23->h3 i24->h3 i25->h3 i26->
h3 i27->h3 i28->h3 i29->h3
##    0.02    0.17    0.26   -0.10   -0.06    0.15    0.11    0.03   -0.07    0.12    0.10    0.
03   -0.03    0.06    0.15
##   b->h4  i1->h4  i2->h4  i3->h4  i4->h4  i5->h4  i6->h4  i7->h4  i8->h4  i9->h4 i10->h4 i11->
h4 i12->h4 i13->h4 i14->h4
##    0.49   -1.59    0.13    0.12    0.11    0.00    0.12    0.14    0.04   -0.26   -0.49    0.
33    0.73   -0.30   -0.21
## i15->h4 i16->h4 i17->h4 i18->h4 i19->h4 i20->h4 i21->h4 i22->h4 i23->h4 i24->h4 i25->h4 i26->
h4 i27->h4 i28->h4 i29->h4
##    1.00    0.21    0.27   -0.11   -0.42    0.13    0.12    0.12    0.10    0.19    0.11   -0.
01    0.21    0.41    0.06
##   b->h5  i1->h5  i2->h5  i3->h5  i4->h5  i5->h5  i6->h5  i7->h5  i8->h5  i9->h5 i10->h5 i11->
h5 i12->h5 i13->h5 i14->h5
##    0.53   -0.05    0.08    0.13    0.30   -0.05    0.06    0.15   -0.07    0.31    0.31   -0.
02   -0.15   -0.02    0.11
## i15->h5 i16->h5 i17->h5 i18->h5 i19->h5 i20->h5 i21->h5 i22->h5 i23->h5 i24->h5 i25->h5 i26->
h5 i27->h5 i28->h5 i29->h5
##   -0.05    0.23    0.24   -0.06    0.09    0.40    0.08    0.18   -0.11    0.31    0.13    0.
19   -0.11    0.14    0.39
##   b->h6  i1->h6  i2->h6  i3->h6  i4->h6  i5->h6  i6->h6  i7->h6  i8->h6  i9->h6 i10->h6 i11->
h6 i12->h6 i13->h6 i14->h6
##    0.07    1.19   -0.07   -0.15    0.21    0.10    0.00   -0.69   -0.23   -0.24   -0.07    0.
74    0.55    0.12    0.54
## i15->h6 i16->h6 i17->h6 i18->h6 i19->h6 i20->h6 i21->h6 i22->h6 i23->h6 i24->h6 i25->h6 i26->
h6 i27->h6 i28->h6 i29->h6
##   -0.40   -0.10   -0.11    0.19   -0.15    0.04   -0.01    0.02    0.03    0.00    0.03   -0.
02    0.05    0.19   -0.12
##   b->h7  i1->h7  i2->h7  i3->h7  i4->h7  i5->h7  i6->h7  i7->h7  i8->h7  i9->h7 i10->h7 i11->
h7 i12->h7 i13->h7 i14->h7
```

```
##   -0.35   -6.16   -0.04   -0.03   -0.07   -0.16   -0.04   -0.13   -0.07   -0.14   -0.17   -0.
02   0.17    0.06   -0.40
## i15->h7 i16->h7 i17->h7 i18->h7 i19->h7 i20->h7 i21->h7 i22->h7 i23->h7 i24->h7 i25->h7 i26->
h7 i27->h7 i28->h7 i29->h7
##    0.27    0.05    0.09   -0.23   -0.18   -0.10    0.00   -0.23   -0.05   -0.09   -0.07   -0.
13   -0.06   -0.07   -0.27
##   b->h8  i1->h8  i2->h8  i3->h8  i4->h8  i5->h8  i6->h8  i7->h8  i8->h8  i9->h8 i10->h8 i11->
h8 i12->h8 i13->h8 i14->h8
##    0.25    0.83   -0.16    0.14    0.14    0.00    0.15    1.02    0.55    0.05   -0.49   -0.
55   -0.34    0.09   -0.86
## i15->h8 i16->h8 i17->h8 i18->h8 i19->h8 i20->h8 i21->h8 i22->h8 i23->h8 i24->h8 i25->h8 i26->
h8 i27->h8 i28->h8 i29->h8
##    0.92    0.42    0.46   -0.38   -0.36    0.06    0.06    0.05    0.07    0.07   -0.01    0.
08    0.10    0.14    0.11
##   b->h9  i1->h9  i2->h9  i3->h9  i4->h9  i5->h9  i6->h9  i7->h9  i8->h9  i9->h9 i10->h9 i11->
h9 i12->h9 i13->h9 i14->h9
##    0.79    0.29   -0.02    0.02    0.57    0.06    0.13    0.13    0.05    0.10    0.30    0.
08    0.15    0.01   -0.03
## i15->h9 i16->h9 i17->h9 i18->h9 i19->h9 i20->h9 i21->h9 i22->h9 i23->h9 i24->h9 i25->h9 i26->
h9 i27->h9 i28->h9 i29->h9
##    0.23    0.16    0.26    0.05    0.08    0.28    0.00    0.11    0.40    0.28    0.03    0.
09    0.40    0.09    0.70
##   b->h10  i1->h10  i2->h10  i3->h10  i4->h10  i5->h10  i6->h10  i7->h10  i8->h10  i9->h10 i10
->h10 i11->h10 i12->h10
##     0.19    -0.75     0.02    -0.04     0.19    -0.04     0.06    -0.04     0.03     0.09    
0.30    -0.08    -0.13
## i13->h10 i14->h10 i15->h10 i16->h10 i17->h10 i18->h10 i19->h10 i20->h10 i21->h10 i22->h10 i23
->h10 i24->h10 i25->h10
##    -0.05    -0.17    -0.13     0.17    -0.05    -0.10     0.49    -0.31    -0.17     0.12    
0.54    -0.31    -0.05
## i26->h10 i27->h10 i28->h10 i29->h10
##     0.13     0.40    -0.04     0.22
##   b->o  h1->o  h2->o  h3->o  h4->o  h5->o  h6->o  h7->o  h8->o  h9->o h10->o
##   0.70  -0.43   1.77  -0.63  -2.36  -0.33   1.03  -4.67   1.84   0.16  -0.56
```

```
#(2B)Below is the reason for what number of hidden neurons would be optimal.
'From the results above, number of hidden neurons when queals to 10 produces the output which ha
s the low MAPE and does not require high computation like when hidden nuerons are 20'
```

```
## [1] "From the results above, number of hidden neurons when queals to 10 produces the output w
hich has the low MAPE and does not require high computation like when hidden nuerons are 20"
```

```
#(2C)
'I would use the optimal neural network from this model since it has lower MAPE when compared to
the MAPE of the linear regression model in the part-1 above.'
```

```
## [1] "I would use the optimal neural network from this model since it has lower MAPE when comp
ared to the MAPE of the linear regression model in the part-1 above."
```

```
#(2D)# Predict the prices of each diamond in the training dataset
train_nn$predicted_price <- predict(nn_Omodel, train_nn)

train_nn$predicted_price <- train_nn$predicted_price * (maxs[2] - mins[2]) + mins[2]
#(1I) # assuming the training dataset is called "diamonds_train"
recommended_diamond <- train_nn %>% mutate(predicted_price = (train_nn$predicted_price)) %>%
  filter(predicted_price <= log(12000)) %>%
  slice_max(Carat.Weight)
recommended_diamond$predicted_price <- exp(recommended_diamond$predicted_price)
# output the details of the recommended diamond
print(recommended_diamond)
```

```
##   Carat.Weight     Price Cut_Fair Cut_Good Cut_Ideal Cut_Signature-Ideal Cut_Very Good Color_
D Color_E Color_F Color_G
## 1    0.4444444 0.4067218        0        0         0                   0             1
0       0       0       0
## 2    0.4444444 0.4130980        0        0         1                   0             0
0       0       0       0
## 3    0.4444444 0.3962564        0        1         0                   0             0
0       0       0       0
##   Color_H Color_I Clarity_FL Clarity_IF Clarity_SI1 Clarity_VS1 Clarity_VS2 Clarity_VVS1 Clar
ity_VVS2 Polish_EX Polish_G
## 1       0       1          0          0           1           0           0            0
0         0        0
## 2       0       1          0          0           1           0           0            0
0         0        0
## 3       0       1          0          0           1           0           0            0
0         0        0
##   Polish_ID Polish_VG Symmetry_EX Symmetry_G Symmetry_ID Symmetry_VG Report_AGSL Report_GIA p
redicted_price
## 1         0         1           0          0           0           1           0          1
11034.20
## 2         0         1           1          0           0           0           0          1
11108.10
## 3         0         1           0          0           0           1           0          1
10697.97
```

```
'For Greg, I will suggest a diamond with 0.486 carat weight, good color, SI1 Clarity, ID symmetr
y and a GIA Report.'
```

```
## [1] "For Greg, I will suggest a diamond with 0.486 carat weight, good color, SI1 Clarity, ID
symmetry and a GIA Report."
```

```
#(2E)
'Building a neural network model using the nnet package in R typically involves the following st
eps:

Data preparation: This involves preparing the data that will be used to train and test the neura
l network. This may include tasks such as data cleaning, normalization or standardization, split
ting the data into training and testing sets, and creating dummy variables for categorical predi
ctors.

Model architecture: The next step is to specify the architecture of the neural network, includin
g the number of hidden layers, the number of neurons in each layer, and the activation functions
used in each layer. The nnet package provides a function called nnet() that allows you to specif
y these parameters.

Training the model: Once the architecture has been defined, the neural network can be trained us
ing the training data. This involves updating the weights and biases of the neurons in the netwo
rk through a process called backpropagation. The nnet() function in nnet package allows you to s
pecify the number of iterations (epochs) to train the neural network.

Model evaluation: After the neural network has been trained, it is important to evaluate its per
formance on the test data. This involves using metrics such as accuracy, precision, recall, and
F1 score to assess how well the model is able to predict outcomes on new, unseen data.

Model refinement: Depending on the performance of the neural network, it may be necessary to ref
ine the model architecture or hyperparameters in order to improve its performance on the test da
ta. This may involve experimenting with different activation functions, increasing the number of
hidden layers or neurons, or changing the learning rate used in backpropagation.

Model deployment: Once the neural network has been trained and evaluated, it can be used to make
predictions on new, unseen data. This may involve integrating the neural network into a larger s
oftware system or deploying it as a standalone application.

Overall, building a neural network model using the nnet package in R requires a solid understand
ing of neural network architecture, backpropagation, and model evaluation, as well as expertise
in data preprocessing and analysis. It is a complex but powerful technique that can be used to s
olve a wide range of problems in fields such as finance, healthcare, and marketing'
```

```
## [1] "Building a neural network model using the nnet package in R typically involves the follo
wing steps:\n\nData preparation: This involves preparing the data that will be used to train and
test the neural network. This may include tasks such as data cleaning, normalization or standard
ization, splitting the data into training and testing sets, and creating dummy variables for cat
egorical predictors.\n\nModel architecture: The next step is to specify the architecture of the
neural network, including the number of hidden layers, the number of neurons in each layer, and
the activation functions used in each layer. The nnet package provides a function called nnet()
that allows you to specify these parameters.\n\nTraining the model: Once the architecture has be
en defined, the neural network can be trained using the training data. This involves updating th
e weights and biases of the neurons in the network through a process called backpropagation. The
nnet() function in nnet package allows you to specify the number of iterations (epochs) to train
the neural network.\n\nModel evaluation: After the neural network has been trained, it is import
ant to evaluate its performance on the test data. This involves using metrics such as accuracy,
precision, recall, and F1 score to assess how well the model is able to predict outcomes on new,
unseen data.\n\nModel refinement: Depending on the performance of the neural network, it may be
necessary to refine the model architecture or hyperparameters in order to improve its performanc
e on the test data. This may involve experimenting with different activation functions, increasi
ng the number of hidden layers or neurons, or changing the learning rate used in backpropagatio
n.\n\nModel deployment: Once the neural network has been trained and evaluated, it can be used t
o make predictions on new, unseen data. This may involve integrating the neural network into a l
arger software system or deploying it as a standalone application.\n\nOverall, building a neural
network model using the nnet package in R requires a solid understanding of neural network archi
tecture, backpropagation, and model evaluation, as well as expertise in data preprocessing and a
nalysis. It is a complex but powerful technique that can be used to solve a wide range of proble
ms in fields such as finance, healthcare, and marketing"
```