Assignment 02

Jawad Adil - 3049429

5/13/2021

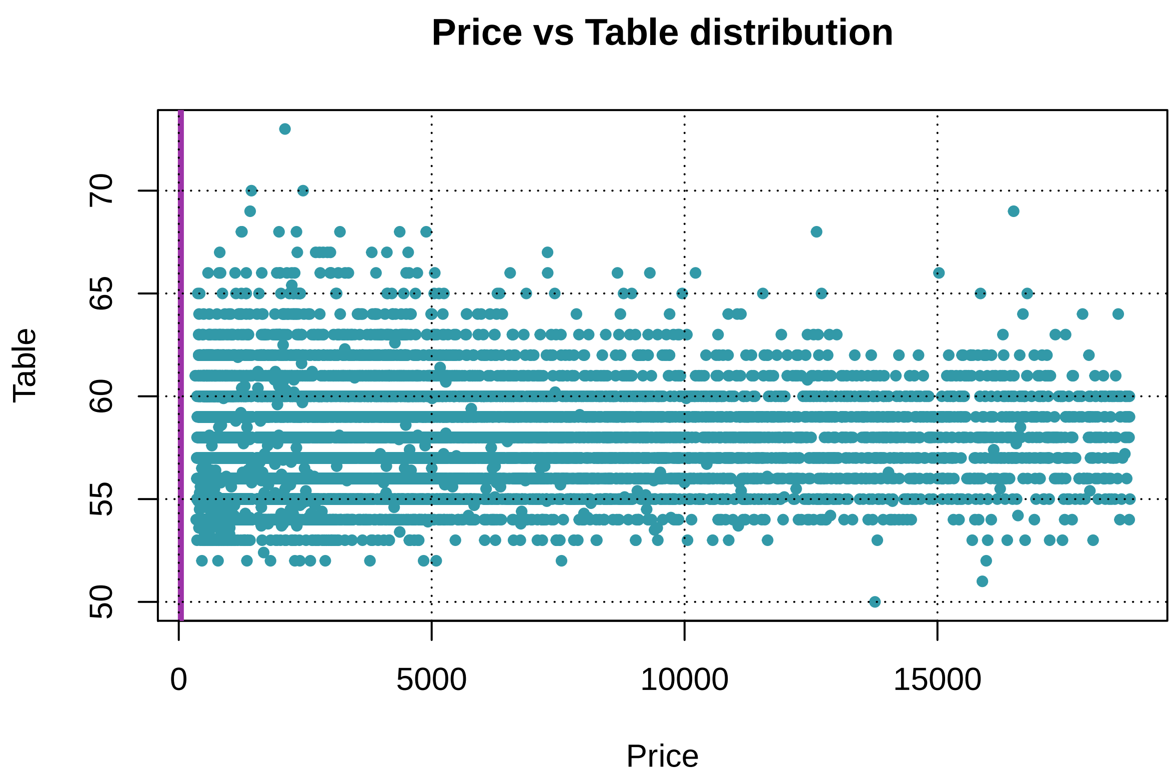
Gathering Data from CSV file and making sure it is in correct form

```
set.seed(1)
# read the data from CSV
DiamondData <- read.csv("C:/Users/jawad adil/Downloads/DiamondDataComplete.csv")

# create a separate sample of 10000 values
s <- sample(nrow(DiamondData), size=15000, replace = FALSE, prob = NULL)
s <- DiamondData[s, ]
# s <- DiamondData

# remove if there's any NA value
s<-na.omit(s)

# replace Very Geod with very Good
s$cut[s$cut == "Very Geod"] <-"Very Good"

# replace higher carat values and make them in range
s$carat<-replace(s$carat,s$carat>5.01,5.01)
```

Task A: Hypothesis testing

Hypothesis 1: Is Diamond price depends upon Diamond table?

```
set.seed(2)
# first plot the graph between price and table
plot(s$price,s$table,xlab = "Price",ylab = "Table",
     main = "Price vs Table distribution",pch=20,col="#3299a8")
# regression line
abline(lm(s$price~s$table),col="#9c32a8",lwd="3")
# printing grid behind
grid(col = "black")
```



H0: Diamond Price is independent of Table attribute of Diamonds

H1: Diamond Price is dependent of Table attribute of Diamonds

Level of Significance: alpha = 0.05

Decision rule:

If p.value is less than the level of significance 0.05 then reject H0 or null hypothesis

Now check for T-Test between price and table attributes

```
set.seed(3)
# calculate T-Test between price and table
t.test(s$price,s$table)
```

```
##
##  Welch Two Sample t-test
##
## data:  s$price and s$table
## t = 119.35, df = 14999, p-value < 2.2e-16
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  3825.298 3953.049
## sample estimates:
##  mean of x  mean of y
## 3946.60320   57.42997
```

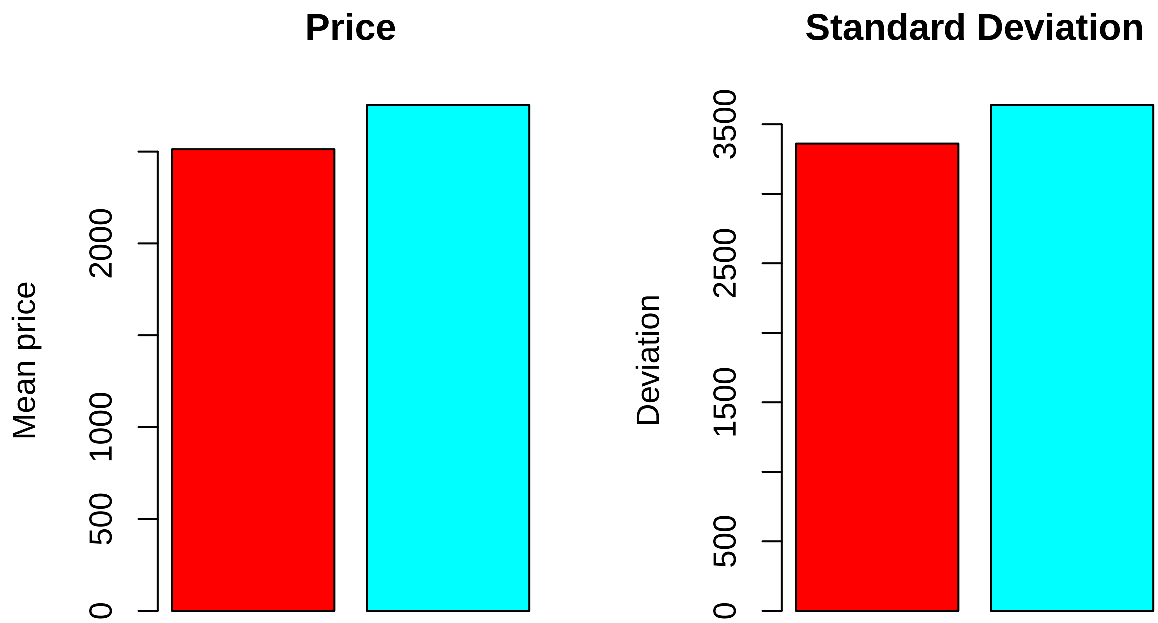
Results:

Since, p.value is less than the 0.05 So, we reject H0. We can conclude that the price of Diamond is not independent of table. It means our hypothesis was false.

Hence, the Price and table for diamond are dependent variables.

Hypothesis 2: Is the price for Diamonds with clarity VVs1 and IF same?

```
set.seed(4)
# separate the diamonds that has clarity = VVS1
VVS1 <- s[which(s$clarity=='VVS1'), ]

# separate the diamonds that has clarity = IF
IF <- s[which(s$clarity=='IF'), ]

# calculating mean prices
priceV <- mean(VVS1$price)
priceI <- mean(IF$price)

# calculating standard deviations
sdV <- sd(VVS1$price)
sdI <- sd(IF$price)

# create vectors for plot
price_values <- c(priceV,priceI)
sd_values <- c(sdV,sdI)

# set 1:2 for plotting side by side
par(mfrow=c(1,2))

# plot price
barplot <- barplot(price_values,col=rainbow(2),ylab = "Mean price",main="Price")

# plot standard deviation
barplot <- barplot(sd_values,col=rainbow(2),ylab = "Deviation",
                   main="Standard Deviation")
```



H0: There is no difference between mean price for diamonds with clarity = VVS1 and clarity = IF.

H1: There is a difference between mean price for diamonds with clarity = VVS1 and clarity = IF.

Level of Significance: alpha = 0.05

Decision rule:

If p.value is less than the level of significance 0.05 then reject H0 or null hypothesis

Apply test on the values to check P-value

```
set.seed(5)
# apply t.test() function to check p-value
test <- t.test(IF$price, VVS1$price)
test
```

```
##
##  Welch Two Sample t-test
##
## data:  IF$price and VVS1$price
```

```
## t = 1.2473, df = 926.58, p-value = 0.2126
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -137.6846  617.8634
## sample estimates:
## mean of x mean of y
##   2752.54   2512.45
```

Results:

Since, p.value is 0.2126177 which is less than the 0.05 So, we reject H0.

Hence, the mean prices for diamonds with clarity VVS1 and IF are different. Mean price for diamonds with clarity VVS1 is 2512.4502868 having standard daviation 3361.3329691 and the mean price for diamonds with clarity IF is 2752.5396825 having standard daviation 3637.4525842.

Task B: Regression and prediction

1: Divide the data into training and test data 75% and 25%

```
# Get the count of 75% rows from training data
training_data_range <- sample(nrow(s), size= floor(.75*nrow(s)), replace = FALSE, prob = NULL)
#s<-as.factor(s)
# Get the 1st 75% rows for training data
training_data <- s[training_data_range, ]

# Get the last 25% rows for testing data
testing_data <- s[-training_data_range, ]

# To confirm we have correctly separated data
# check if the starting values are same or not
head(training_data)
```

```
##       carat       cut color clarity depth table price    x    y    z
## 8522   0.90      Good     H     SI1  59.0    62  3577 6.23 6.27 3.69
## 24117  1.50   Premium     G     VS2  62.5    59 12787 7.27 7.23 4.53
## 49977  0.59     Ideal     F     VS2  60.6    56  1834 5.43 5.48 3.31
## 8678   0.30 Very Good     G     VS2  63.5    55   675 4.25 4.22 2.69
## 47828  0.41   Premium     J    VVS1  62.2    59   775 4.71 4.75 2.94
## 18002  0.30     Ideal     D     VS2  60.8    57   911 4.34 4.31 2.63
```

```
# similarly check the starting 6 values of testing data as well
head(testing_data)
```

```
##       carat       cut color clarity depth table price    x    y    z
## 43307  1.00   Premium     I     SI1  62.2    62  3360 6.39 6.33 3.96
## 39294  0.31   Premium     E    VVS1  61.8    59  1012 4.33 4.31 2.67
## 43809  1.01   Premium     E     VS2  60.8    58  4706 6.43 6.40 3.90
## 7075   0.35     Ideal     G     VS1  60.5    57   906 4.58 4.55 2.76
## 21784  1.04     Ideal     F    VVS2  61.2    57  9169 6.53 6.50 3.99
## 39645  1.00 Very Good     G    VVS2  62.1    59  7242 6.34 6.45 3.97
```

Data in both sets are different, Hence we can confirm that data is correctly splitted.

```
# total number of rows
nrow(s)
```

```
## [1] 15000
```

```
# check the number of rows in training data
nrow(training_data)
```

```
## [1] 11250
```

```r
#check the number of rows in testing data
nrow(testing_data)
```

```
## [1] 3750
```

This also indicates that our data is correctly separated.

Now Create a linear model and test it using prediction function

```r
set.seed(6)

# create a linear model
linear_model <- lm(price~.,data=training_data)

# check the cofficients of linear model
coef(linear_model)
```

```
##  (Intercept)        carat      cutGood     cutIdeal   cutPremium cutVery Good
## -22288.54348  11248.39555    415.34800    686.77437    639.08863    534.18838
##       colorE       colorF       colorG       colorH       colorI       colorJ
##   -190.04544   -242.95956   -460.23621   -953.43262  -1436.28891  -2312.71879
##    clarityIF   claritySI1   claritySI2   clarityVS1   clarityVS2  clarityVVS1
##   5310.81463   3724.98703   2763.66648   4701.09526   4339.85941   5093.33713
##  clarityVVS2        depth        table            x            y            z
##   4984.62326    320.63229    -12.36195    214.47601   2743.16212  -6464.16145
```

```r
# predict the prices for testing data
prediction <- predict(linear_model,testing_data)

# find correlation between prediction and prices
correlation <- cor(prediction,testing_data$price)
correlation
```

```
## [1] 0.9044415
```

```r
# find cor squared which is equal to R^2
r_squared <- cor(prediction,testing_data$price)^2
r_squared
```

```
## [1] 0.8180145
```

```r
# find adjusted r square using summary function
adjusted_r_squared <- summary(linear_model)$adj.r.squared
adjusted_r_squared
```

```
## [1] 0.9174407
```

```
# find the RMSE using caret library function to verify
rmse <- RMSE(testing_data$price,prediction)
rmse
```

```
## [1] 1801.877
```

Now normalize the data and check the linear regression again

```
# set random seed
set.seed(7)
# function to normalize the values
normalize <- function(x) {
  return ((x - min(x)) / (max(x) - min(x)))
}

#create a copy to normalize the data, actual dataset will be preserved
s1<-s
s1$cut <- as.numeric(as.factor(s1$cut))
s1$clarity <- as.numeric(as.factor(s1$clarity))
s1$color <- as.numeric(as.factor(s1$color))

# normalize the data
s1 <- as.data.frame(lapply(s1, normalize))

#defining range of training data
normalized_training_data_range <- sample(nrow(s1), size= floor(.75*nrow(s)), replace = FALSE,
                                         prob = NULL)

# Get the 1st 75% rows for training data
normalized_training_data <- s1[normalized_training_data_range, ]

# Get the last 25% rows for testing data
normalized_testing_data <- s1[-normalized_training_data_range, ]

# create linear model for normalized data
normalized_linear_model <- lm(price~.,data=normalized_training_data)

# check the coefficients of linear model
coef(normalized_linear_model)
```

```
## (Intercept)       carat         cut       color     clarity       depth
##  0.27446946  2.13202180  0.01669382 -0.08527543  0.11514392 -0.22846790
##       table           x           y           z
## -0.10332313 -0.56384378  0.16231373  0.02774017
```

```
# predict the prices for testing data
normalized_prediction <- predict(normalized_linear_model,normalized_testing_data)

# find correlation between prediction and prices
normalized_correlation <- cor(normalized_prediction,normalized_testing_data$price)
normalized_correlation
```

```
## [1] 0.9416669
```

```
# find cor squared which is equal to R^2
normalized_r_square<- cor(normalized_prediction,normalized_testing_data$price)^2
normalized_r_square
```

```
## [1] 0.8867365
```

```
# find adjusted r square using summary function
normalized_adjusted_r_squared<- summary(normalized_linear_model)$adj.r.squared
normalized_adjusted_r_squared
```

```
## [1] 0.8796275
```

```
# find the RMSE using caret library function to verify
normalized_rmse <- RMSE(normalized_testing_data$price,normalized_prediction)
normalized_rmse
```

```
## [1] 0.07316947
```

Conclusion:

we can compare our results by comparing them side by side.

```
                        Simple Data           Normalized Data
```

Correlation: =========== 0.9044415 ========= 0.9416669

R-Squared: ============0.8180145 ========= 0.8867365

Adjusted-R-Squared: =======0.9174407 ========= 0.8796275

Root mean square error: ====1801.877497 ======= 0.0731695

So the better performance achieved is : 1801.877497

Task C: Classifications and prediction

```
# set random seed value
set.seed(8)

# splitting 80/20% training a testing data
trainingInd <- createDataPartition(s$cut, p= 0.8, list = F)
training_data <- s[trainingInd,]
test_data <- s[-trainingInd,]

# setting control parameters for training
trainctrl <- trainControl(method = "repeatedcv", number = 10, repeats = 3)

# training kNN model
knn_fit <- train(cut ~., data = training_data, method = "knn",trControl = trainctrl,tuneLength = 10)

# predicting the cut classes
knnPredict <- predict(knn_fit, newdata = test_data )

# confusion matrix to find accuracy and other related stuff
knn_con <- confusionMatrix(knnPredict, as.factor(test_data$cut))
knn_con
```

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction  Fair Good Ideal Premium Very Good
##   Fair         5    4     3       5         3
##   Good         8   27    28      16        29
##   Ideal       33   96   875     243       285
##   Premium     24   77   154     345       190
##   Very Good   20   65   159     150       153
##
## Overall Statistics
##
##                Accuracy : 0.4688
##                  95% CI : (0.4508, 0.4869)
##     No Information Rate : 0.4067
##     P-Value [Acc > NIR] : 3.598e-12
##
##                   Kappa : 0.2208
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: Fair Class: Good Class: Ideal Class: Premium
## Sensitivity             0.055556    0.100372       0.7178         0.4545
## Specificity             0.994840    0.970308       0.6305         0.8012
## Pos Pred Value          0.250000    0.250000       0.5711         0.4367
## Neg Pred Value          0.971448    0.916234       0.7652         0.8124
## Prevalence              0.030030    0.089756       0.4067         0.2533
## Detection Rate          0.001668    0.009009       0.2920         0.1151
```

```
## Detection Prevalence    0.006673    0.036036       0.5112          0.2636
## Balanced Accuracy       0.525198    0.535340       0.6741          0.6279
##                      Class: Very Good
## Sensitivity                   0.23182
## Specificity                   0.83141
## Pos Pred Value                0.27971
## Neg Pred Value                0.79306
## Prevalence                    0.22022
## Detection Rate                0.05105
## Detection Prevalence          0.18252
## Balanced Accuracy             0.53161
```

```r
knn_accuracy <- knn_con$overall['Accuracy']
knn_accuracy
```

```
##  Accuracy
## 0.4688021
```

```r
# set random seed
set.seed(9)

# split training and testing data with ratio 80/20 %
trainingInd <- createDataPartition(s$cut, p= 0.8, list = F)
training_data <- s[trainingInd,]
test_data <- s[-trainingInd,]

# train using C5.0 trees methods
C5_fit <- train(cut~., data = training_data, method = "C5.0")

# predicting cut classes using the C5 fit
C5_predict <- predict(C5_fit, newdata= test_data )

# calculating confusion matrix to get the accuracy and stuff
c5_con <- confusionMatrix(C5_predict, as.factor(test_data$cut))
c5_con
```

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction  Fair Good Ideal Premium Very Good
##   Fair        76    8     2       0         2
##   Good         6  173     0       4        37
##   Ideal        2    4  1126      85       161
##   Premium      5   28    55     614       167
##   Very Good    1   56    36      56       293
##
## Overall Statistics
##
##                Accuracy : 0.7614
##                  95% CI : (0.7458, 0.7766)
##     No Information Rate : 0.4067
##     P-Value [Acc > NIR] : < 2.2e-16
##
```

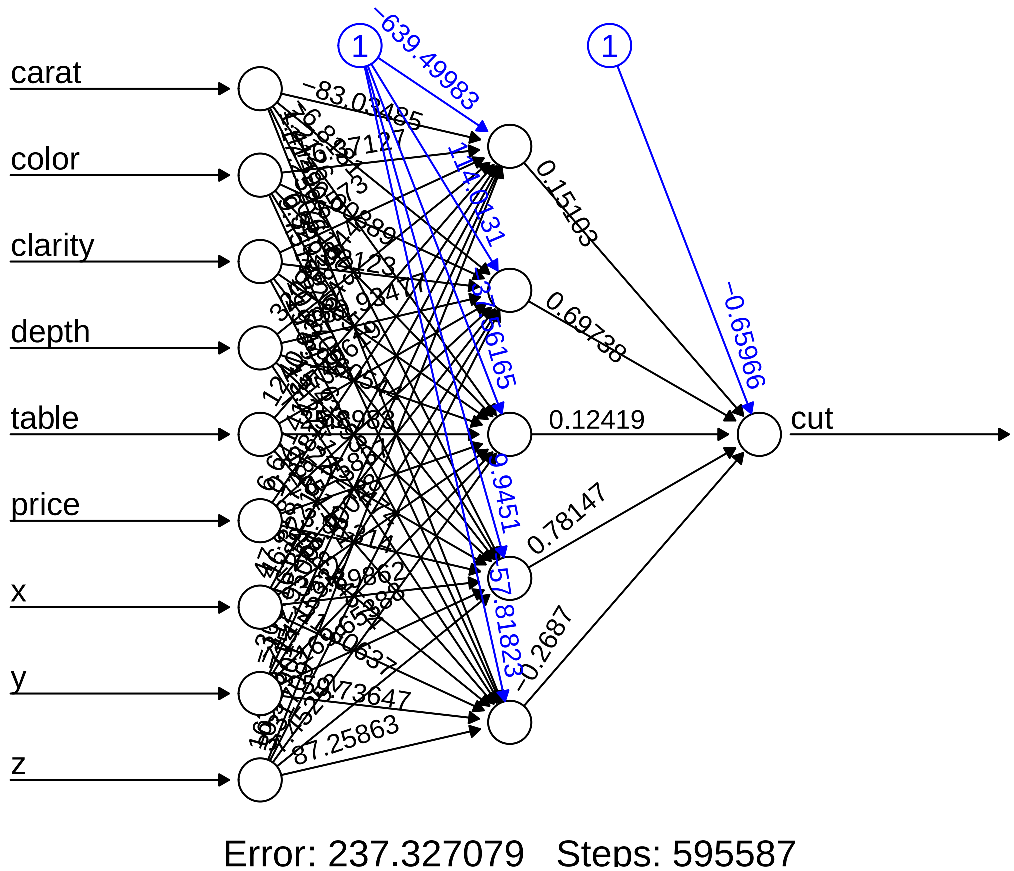
```
##                    Kappa : 0.659
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: Fair Class: Good Class: Ideal Class: Premium
## Sensitivity              0.84444     0.64312       0.9237         0.8090
## Specificity              0.99587     0.98277       0.8583         0.8861
## Pos Pred Value           0.86364     0.78636       0.8171         0.7066
## Neg Pred Value           0.99519     0.96543       0.9426         0.9319
## Prevalence               0.03003     0.08976       0.4067         0.2533
## Detection Rate           0.02536     0.05772       0.3757         0.2049
## Detection Prevalence     0.02936     0.07341       0.4598         0.2900
## Balanced Accuracy        0.92016     0.81295       0.8910         0.8475
##                      Class: Very Good
## Sensitivity                   0.44394
## Specificity                   0.93624
## Pos Pred Value                0.66290
## Neg Pred Value                0.85636
## Prevalence                    0.22022
## Detection Rate                0.09776
## Detection Prevalence          0.14748
## Balanced Accuracy             0.69009
```

```r
c5_accuracy <- c5_con$overall['Accuracy']
c5_accuracy
```

```
##  Accuracy
## 0.7614281
```

```r
# setting random seed
set.seed(10)

normalize <- function(x) {
  return ((x - min(x)) / (max(x) - min(x)))
}

# converting strings to numerics in the dataset
s$cut <- as.numeric(as.factor(s$cut))
s$clarity <- as.numeric(as.factor(s$clarity))
s$color <- as.numeric(as.factor(s$color))

# normalize the data
s <- as.data.frame(lapply(s, normalize))

# split the data into 80/20 training and testing+ data
trainingInd <- createDataPartition(s$cut, p= 0.8, list = F)
training_data <- s[trainingInd,]
test_data <- s[-trainingInd,]

# using neuralnet function to create ANN fit
ANN_fit <- neuralnet(cut~., data = training_data, hidden = 5,stepmax=1e6)
```

```r
# ploting the neural network nodes with weights
plot(ANN_fit,rep = "best")
```




```r
# Computing results with all columns other than cuts
ANN_results <- compute(ANN_fit, test_data[,-2])

# getting prediction
predicted_strength <- ANN_results$net.result

# finding correlation
ANN_accuracy <- cor(predicted_strength, test_data$cut)
ANN_accuracy
```

```
##           [,1]
## [1,] 0.5912368
```

Accuracy of all of the above methods is given below

KNN: 0.4688021

C5.0: 0.7614281

ANN: 0.5912368

So, The Maximum Accuracy achieved from above 3 models is: 0.7614281.