Diamonds Price Prediction by Its Characteristics

Inga Aritenco

25/10/2021

INTRODUCTION

Day by day the world of information technology grows and the stream of data expands. With this, Data Science has emerged as a top-of-the-line specialty. Professionals in the field not only wrangle, analyze and visualize the information, they also use machine learning approaches in order to process a modern treasure trove – data.

Statistical learning, which can be either machine learning or deep learning (the last includes artificial neural networks) use various algorithms to reveal trends or produce insights to make better decisions for data-driven companies. Examples of the universal application of these methods in the Artificial Intelligence subfields are face detection, speech recognition or stock price prediction.

Machine learning approaches are helpful to assess the importance of stocks in the corporate and financial markets. As an example, it can be useful for cryptocurrency traders and high-value metals (gold, silver, platinum) and precious gemstones(sapphire, ruby, diamond) retailers.

This report uses Diamonds dataset downloaded from Kaggle[1], a platform for Data Scientists. The dataset was provided by Tiffany & Co.[2](an American luxury jewelry and specialty retailer headquartered in New York City) snapshot pricelist from 2017.

The main task of the project is to discover the correlation between diamonds attributes and to predict prices by its characteristics. In order to achieve the goal should be used machine learning algorithms, such as `Decision Tree` and `Random Forest`. The Diamonds dataset will be separated in two subsets: one for training, which will be divided by 80% and another one for testing(in other words, validation), which will be split by 20% respectively. Also, will be used the `Confusion Matrix` to compute accuracy of the models. As an addendum to the project will be covered a neural network model.

Exploratory Data Analysis comprises the next sections:

1. Introduction
2. Analysis | Data Preparation | Data Exploration | Data Preprocessing | Visualization
3. Data Modeling and Results
4. Conclusion

ANALYSIS

Data Preparation

Before to start working with Diamonds dataset, download all required packages, which will be used throughout data analysis. If pre-installed, just make use of the libraries.

[1]https://www.kaggle.com/shivam2503/diamonds
[2]https://www.tiffany.com/

```r
# Download all required packages:
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(dplyr)) install.packages("dplyr", repos = "http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(randomForest)) install.packages("randomForest", repos = "http://cran.us.r-project.org")
if(!require(neuralnet)) install.packages("neuralnet", repos = "http://cran.us.r-project.org")
if(!require(lattice)) install.packages("lattice", repos = "http://cran.us.r-project.org")
if(!require(ggplot2)) install.packages("ggplot2", repos = "http://cran.us.r-project.org")
if(!require(GGally)) install.packages("GGally", repos = "http://cran.us.r-project.org")
if(!require(RColorBrewer)) install.packages("RColorBrewer", repos = "http://cran.us.r-project.org")
if(!require(varhandle)) install.packages("varhandle", repos = "http://cran.us.r-project.org")
if(!require(gridExtra)) install.packages("gridExtra", repos = "http://cran.us.r-project.org")
if(!require(tinytex)) install.packages("tinytex", repos = "http://cran.us.r-project.org")
if(!require(knitr)) install.packages("knitr", repos = "http://cran.us.r-project.org")
# If all the packages are pre-installed...
library(tidyverse)
library(dplyr)
library(caret)
library(randomForest)
library(neuralnet)
library(lattice)
library(ggplot2)
library(GGally)
library(RColorBrewer)
library(varhandle) # for converting categorical data into binary
library(gridExtra)
library(tinytex)
library(knitr)
```

Afterwards, download the Diamonds dataset with the following code:

```r
# Download CSV file from author's github account:
# https://raw.githubusercontent.com/oryme/diamonds-project/main/diamonds.csv
# The dataset was originally taken from Kaggle Platform for Data Scientists
# https://www.kaggle.com/shivam2503/diamonds

diamonds <- read.csv(file =
                       "https://raw.githubusercontent.com/oryme/diamonds-project/main/diamonds.csv")
```

Data Exploration

At the beginning, explore data dimensions. It consists of 53,940 observations and 11 variables. There are `3 categorical` variables in the Diamonds dataset:

`cut` (proportions, symmetry, and polish) is a measure of how a diamond's facets interact with light;

`color` comes in a variety (it can be pink, blue, even yellow). The less body color in a white diamond, the more true color it will reflect, and thus the greater its value;

`clarity` grades assess the number, size, relief, and position of inclusions and blemishes;

and `7 continuous`:

`carat` is the original unit of weight/measure for diamond traders. 1 carat ~ 0,2 grams. As the carat size of a diamond increases, the diamond's price increases at an increasing rate;

`depth` is an aspect of a cut, it affects on how much light is reflected back to an observer;

`table` is the flat surface on the very top of the stone that resembles an actual table top. As the largest facet of a diamond, the table reflects and refracts light, making the diamond more brilliant;

`x` length of a diamond in millimeters, from 0 to 10.74;

`y` width of a diamond in mm, from 0 to 58.9;

`z` depth of a diamond in mm, from 0 to 31.8;

`price` in US dollars, from `$326` to `$18,823`.

To improve productivity of subsequent data processing for implementing machine learning algorithms, the categorical values (`cut`, `color`, `clarity`) should be converted from `character` vector to `as.factor`. The Diamonds dataframe was checked for missing values and such kind of values were not detected. At this stage, this means that we should not clean the data.

```
# Data dimensions
dim(diamonds)
```

```
## [1] 53940    11
```

```
# First look at `diamonds` dataset
str(diamonds)
```

```
## 'data.frame':    53940 obs. of  11 variables:
##  $ X      : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ carat  : num  0.23 0.21 0.23 0.29 0.31 0.24 0.24 0.26 0.22 0.23 ...
##  $ cut    : chr  "Ideal" "Premium" "Good" "Premium" ...
##  $ color  : chr  "E" "E" "E" "I" ...
##  $ clarity: chr  "SI2" "SI1" "VS1" "VS2" ...
##  $ depth  : num  61.5 59.8 56.9 62.4 63.3 62.8 62.3 61.9 65.1 59.4 ...
##  $ table  : num  55 61 65 58 58 57 57 55 61 61 ...
##  $ price  : int  326 326 327 334 335 336 336 337 337 338 ...
##  $ x      : num  3.95 3.89 4.05 4.2 4.34 3.94 3.95 4.07 3.87 4 ...
##  $ y      : num  3.98 3.84 4.07 4.23 4.35 3.96 3.98 4.11 3.78 4.05 ...
##  $ z      : num  2.43 2.31 2.31 2.63 2.75 2.48 2.47 2.53 2.49 2.39 ...
```

```
## For commodity purposes in future data analysis convert categorical values
# `cut`, `color`, `clarity` to `as.factor`
diamonds <- mutate(diamonds,
                   cut = as.factor(cut),
                   color = as.factor(color),
                   clarity = as.factor(clarity))

# First five rows of the dataset
head(diamonds)
```

```
##   X carat       cut color clarity depth table price    x    y    z
## 1 1  0.23     Ideal     E     SI2  61.5    55   326 3.95 3.98 2.43
## 2 2  0.21   Premium     E     SI1  59.8    61   326 3.89 3.84 2.31
## 3 3  0.23      Good     E     VS1  56.9    65   327 4.05 4.07 2.31
## 4 4  0.29   Premium     I     VS2  62.4    58   334 4.20 4.23 2.63
## 5 5  0.31      Good     J     SI2  63.3    58   335 4.34 4.35 2.75
## 6 6  0.24 Very Good     J    VVS2  62.8    57   336 3.94 3.96 2.48
```

```
# Make sure if there are any missing values in the dataframe
sum(is.na(diamonds))
```

```
## [1] 0
```

Were observed that each categorical Diamonds attribute has several occurrences. To understand it more in-depth here is a brief explanation:

1. Each cut has five grades of quality:

`Fair` cut Diamonds offer little brilliance, as light easily exits through the bottom and sides of the diamond. It may be a satisfactory choice for smaller carats and those acting as side stones. The number of `Fair` cuts in the dataset is `1610`.

`Good` cut Diamonds showcase brilliance and sparkle, with much of the light reflecting through the table to the viewer's eye. The number of `Good` cuts in observed dataset is `4906`.

`Very Good` cut Diamonds offer exceptional brilliance and fire. A large majority of the entering light reflects through the diamond's table. Th number of `Very Good` cuts in the dataset is `12,082`.

`Ideal` cut Diamonds reflects nearly all the light that enters it through the table (top) and crown (upper section) of the stone. The number and arrangement of facets allows superb refraction of light resulting in unparalleled brilliance and fire. (To maximize reflected light, an ideal cut round brilliant diamond has exactly 58 facets.) The number of `Ideal` cuts in the dataset is `21,551`.

`Premium` demonstrates subtle variations from the `Ideal`. The number of observations in the dataset is `13,791`.

```
# Count the number of occurrences for each diamonds attribute for `cut`
table(diamonds$cut)
```

```
##
##      Fair      Good     Ideal   Premium Very Good
##      1610      4906     21551     13791     12082
```

2. Each diamond with `clarity` factor has 8 grades, which are represented thought the below script.

`I1` (`Included`) inclusions are obvious under 10x magnification which may affect transparency and brilliance;

`IF` (`Internally Flawless`) no inclusions visible under 10x magnification;

`SI1` and `SI2` (`Slightly Included`) inclusions are noticeable under 10x magnification;

`VS1` and `VS2` (`Very Slightly Included`) inclusions are observed with effort under 10x magnification, but can be characterized as minor;

`VVS1` and `VVS2` (`Very, Very Slightly Included`) so slight inclusions are difficult for a skilled grader to see under 10x magnification;

```
# Count the number of occurrences for each diamonds attribute for `clarity`
table(diamonds$clarity)
```

```
##
##    I1    IF   SI1   SI2   VS1   VS2  VVS1  VVS2
##   741  1790 13065  9194  8171 12258  3655  5066
```

3. Each `color` of a diamond has 7 grades:

`D` > `E` > `F` (`Colorless`) there are differences between these colors. They can be detected only by a gemologist in side by side comparisons, and rarely by the untrained eye.

`G` > `H` > `I` > `J` (`Near Colorless`) diamonds are suitable for a platinum or white gold setting, which would normally betray any hint of color in a diamond.

```
# Count the number of occurrences for each diamonds attribute for `color`
table(diamonds$color)
```

```
##
##     D     E     F     G     H     I     J
##  6775  9797  9542 11292  8304  5422  2808
```

Data Preprocessing

In Diamonds data was noticed a column `X`, which refers to each variable in order. The decision was to remove it, because it does not affects much on the process of data analysis.

```
# Exclude unnecessary column `X`
diamonds <- subset(diamonds, select = -c(X))
# Check how the data looks like after excluding `X` column
head(diamonds)
```

```
##   carat       cut color clarity depth table price    x    y    z
## 1  0.23     Ideal     E     SI2  61.5    55   326 3.95 3.98 2.43
## 2  0.21   Premium     E     SI1  59.8    61   326 3.89 3.84 2.31
## 3  0.23      Good     E     VS1  56.9    65   327 4.05 4.07 2.31
## 4  0.29   Premium     I     VS2  62.4    58   334 4.20 4.23 2.63
## 5  0.31      Good     J     SI2  63.3    58   335 4.34 4.35 2.75
## 6  0.24 Very Good     J    VVS2  62.8    57   336 3.94 3.96 2.48
```

The summary statistics of the entire dataset shows minimum, maximum and average value for each diamonds quality factor. Noticeably, `length`(x), `width`(y) and `depth`(z) of a diamond have zero values. It can lead the outliers of the data, which can show the wrong observations in future visualization.

```
summary(diamonds)
```

```
##      carat               cut        color       clarity          depth      
##  Min.   :0.2000   Fair     : 1610   D: 6775   SI1    :13065   Min.   :43.00  
##  1st Qu.:0.4000   Good     : 4906   E: 9797   VS2    :12258   1st Qu.:61.00  
##  Median :0.7000   Ideal    :21551   F: 9542   SI2    : 9194   Median :61.80  
##  Mean   :0.7979   Premium  :13791   G:11292   VS1    : 8171   Mean   :61.75  
##  3rd Qu.:1.0400   Very Good:12082   H: 8304   VVS2   : 5066   3rd Qu.:62.50  
##  Max.   :5.0100                     I: 5422   VVS1   : 3655   Max.   :79.00  
##                                     J: 2808   (Other): 2531                  
##      table           price             x                y         
##  Min.   :43.00   Min.   :  326   Min.   : 0.000   Min.   : 0.000  
##  1st Qu.:56.00   1st Qu.:  950   1st Qu.: 4.710   1st Qu.: 4.720  
##  Median :57.00   Median : 2401   Median : 5.700   Median : 5.710  
##  Mean   :57.46   Mean   : 3933   Mean   : 5.731   Mean   : 5.735  
##  3rd Qu.:59.00   3rd Qu.: 5324   3rd Qu.: 6.540   3rd Qu.: 6.540  
##  Max.   :95.00   Max.   :18823   Max.   :10.740   Max.   :58.900  
##                                                                   
##        z         
##  Min.   : 0.000  
##  1st Qu.: 2.910  
##  Median : 3.530  
##  Mean   : 3.539  
##  3rd Qu.: 4.040  
##  Max.   :31.800  
## 
```

To avoid this, let's use the following code:

```
# Filter each value and equalize it to zero
filter(diamonds, x == 0 | y == 0 | z == 0)
```

```
##    carat       cut color clarity depth table price    x    y z
## 1   1.00   Premium     G     SI2  59.1    59  3142 6.55 6.48 0
## 2   1.01   Premium     H      I1  58.1    59  3167 6.66 6.60 0
## 3   1.10   Premium     G     SI2  63.0    59  3696 6.50 6.47 0
## 4   1.01   Premium     F      SI2  59.2    58  3837 6.50 6.47 0
## 5   1.50      Good     G       I1  64.0    61  4731 7.15 7.04 0
## 6   1.07     Ideal     F      SI2  61.6    56  4954 0.00 6.62 0
## 7   1.00 Very Good     H      VS2  63.3    53  5139 0.00 0.00 0
## 8   1.15     Ideal     G      VS2  59.2    56  5564 6.88 6.83 0
## 9   1.14      Fair     G      VS1  57.5    67  6381 0.00 0.00 0
## 10  2.18   Premium     H      SI2  59.4    61 12631 8.49 8.45 0
## 11  1.56     Ideal     G      VS2  62.2    54 12800 0.00 0.00 0
## 12  2.25   Premium     I      SI1  61.3    58 15397 8.52 8.42 0
## 13  1.20   Premium     D     VVS1  62.1    59 15686 0.00 0.00 0
## 14  2.20   Premium     H      SI1  61.2    59 17265 8.42 8.37 0
## 15  2.25   Premium     H      SI2  62.8    59 18034 0.00 0.00 0
## 16  2.02   Premium     H      VS2  62.7    53 18207 8.02 7.95 0
## 17  2.80      Good     G      SI2  63.8    58 18788 8.90 8.85 0
## 18  0.71      Good     F      SI2  64.1    60  2130 0.00 0.00 0
## 19  0.71      Good     F      SI2  64.1    60  2130 0.00 0.00 0
## 20  1.12   Premium     G       I1  60.4    59  2383 6.71 6.67 0
```

```
# Drop null observations
diamonds <- subset(diamonds, x!=0 & y!=0 & z!=0)
```

Until we start to visualize Diamonds data, find out the number of diamonds by its price.

Less than $500: `1729`;

Less than $250: there is no such kind of diamonds;

More or equals to $15,000: `1650`.

```
# Amount of diamonds that cost less than $500
summary(diamonds$price < 500)
```

```
##    Mode   FALSE    TRUE
## logical   52191    1729
```

```
# Amount of diamonds that cost less than $250
summary(diamonds$price < 250)
```

```
##    Mode   FALSE
## logical   53920
```

```
# Amount of diamonds that cost $15,000 or more
summary(diamonds$price >= 15000)
```

```
##    Mode   FALSE    TRUE
## logical   52270    1650
```

There is only one the most expensive `Premium` diamond in the dataset which is `Near Colorless` and costs `$18,823`.

There are also two low-cost diamonds with the same price (`$326`) and color (`Colorless`), but different grades of `cut` and `clarity`: the fist one is `Ideal` and the second one is `Premium`.

```
# The most expensive diamond in the dataset
subset(diamonds, price == max(price))
```

```
##       carat     cut color clarity depth table price   x    y    z
## 27750  2.29 Premium     I     VS2  60.8    60 18823 8.5 8.47 5.16
```

```
# The most low-cost diamond
subset(diamonds, price == min(price))
```

```
##   carat     cut color clarity depth table price    x    y    z
## 1  0.23   Ideal     E     SI2  61.5    55   326 3.95 3.98 2.43
## 2  0.21 Premium     E     SI1  59.8    61   326 3.89 3.84 2.31
```

Data Visualization

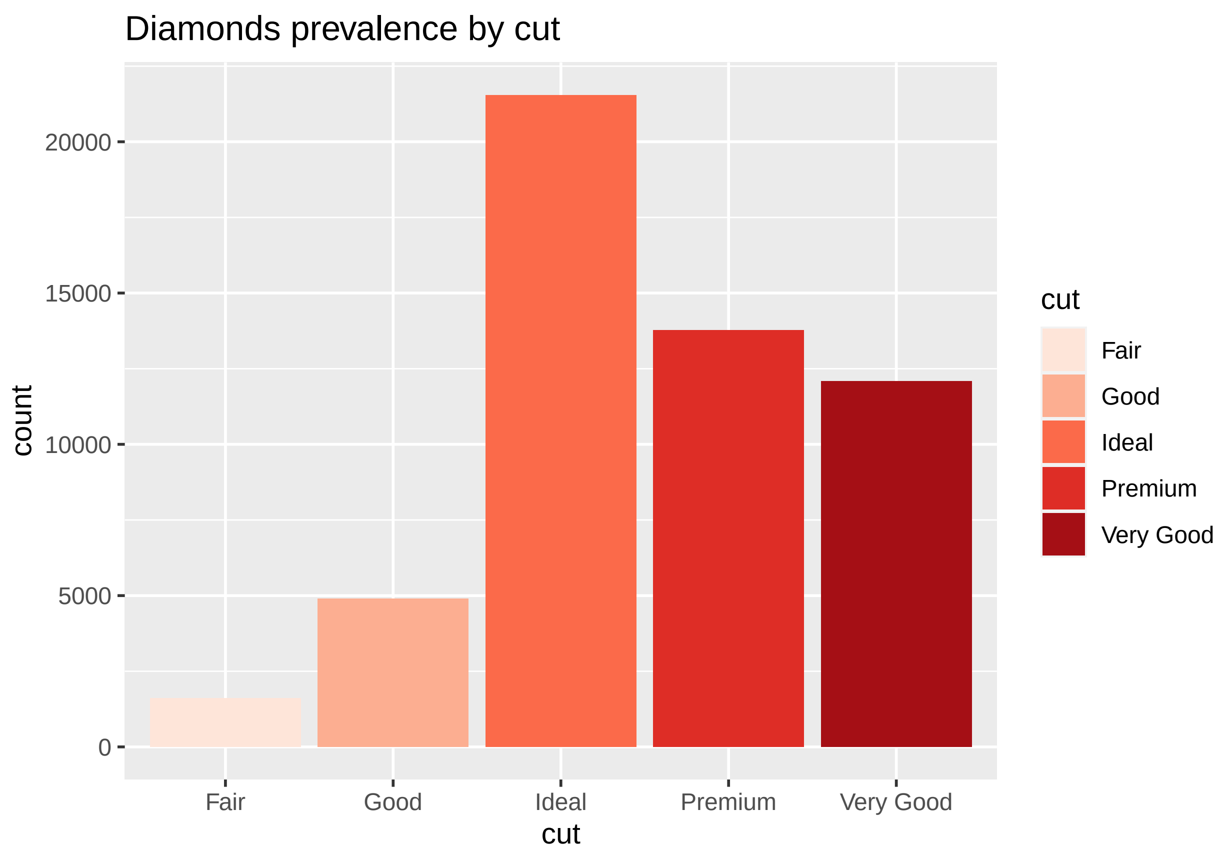
Firstly, let's analyze `categorical` features of the Diamonds dataset. Create bar plots to discover diamonds prevalence by `cut`, `clarity` and `color`.

As shown below, the largest number of diamonds is located in `Ideal` quality category.

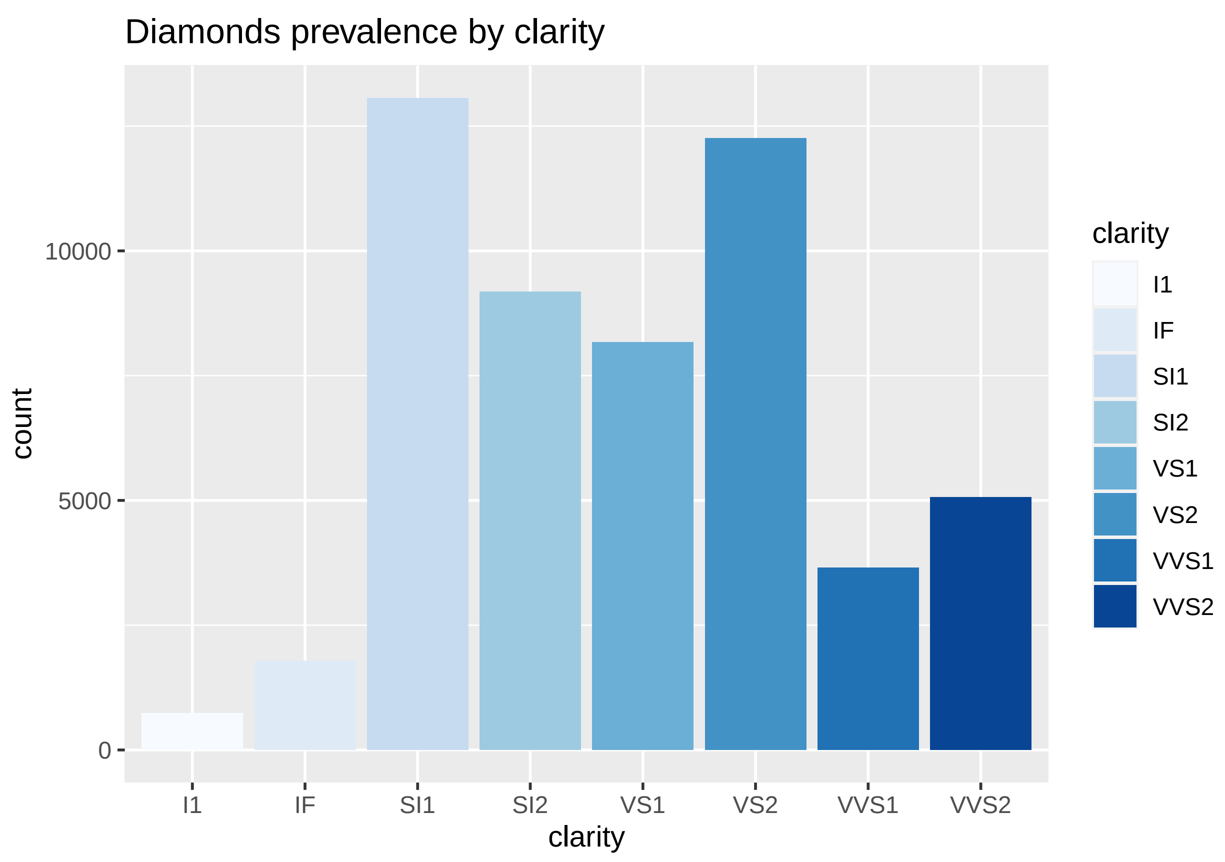
The most prevalent grade in `clarity` is `SI1`, which means the inclusions can be noticed under 10x magnification.

The most common diamonds `color` is represented by a letter `G`(`Near Colorless`).

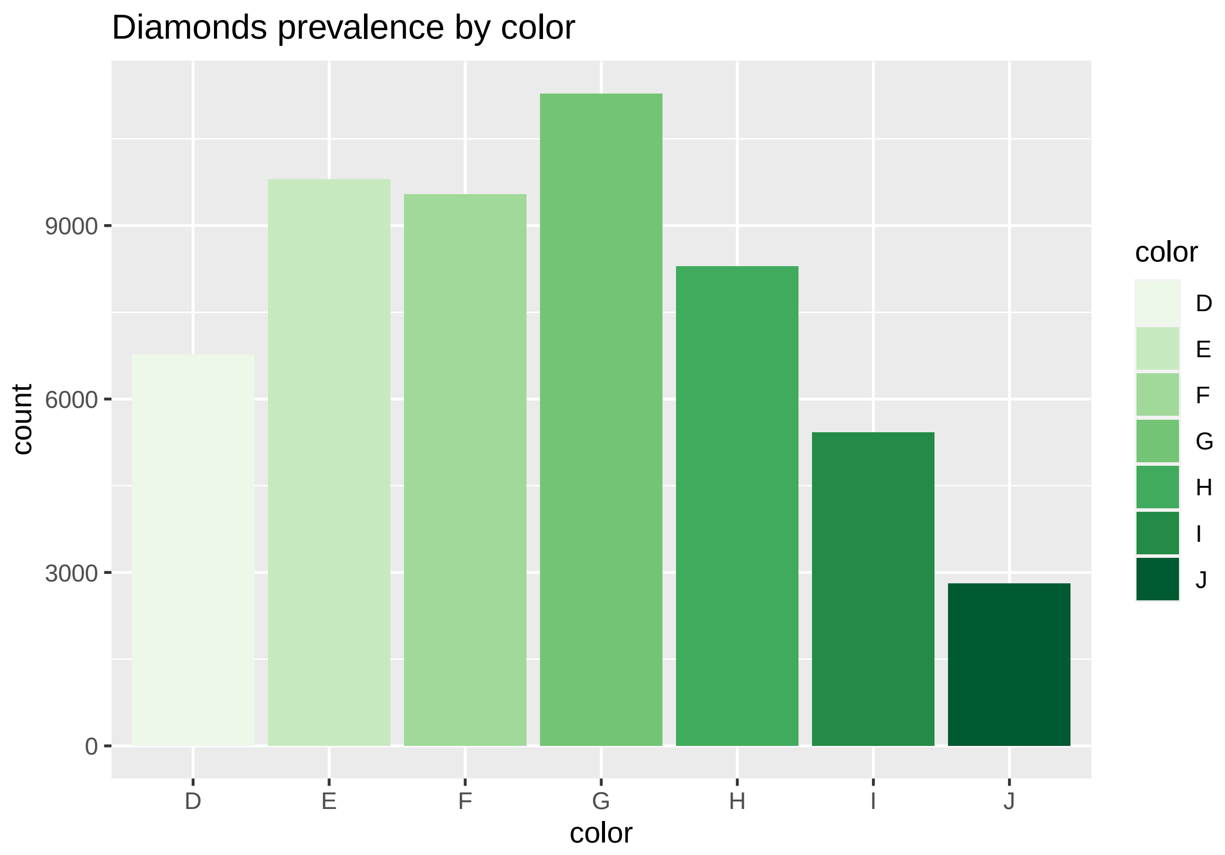
```
ggplot(diamonds) +
  geom_bar(aes(cut, fill = cut)) +
  labs(title = "Diamonds prevalence by cut") +
  scale_fill_brewer(palette = "Reds")
```



```
ggplot(diamonds) +
  geom_bar(aes(clarity, fill = clarity)) +
  labs(title = "Diamonds prevalence by clarity") +
  scale_fill_brewer(palette = "Blues")
```

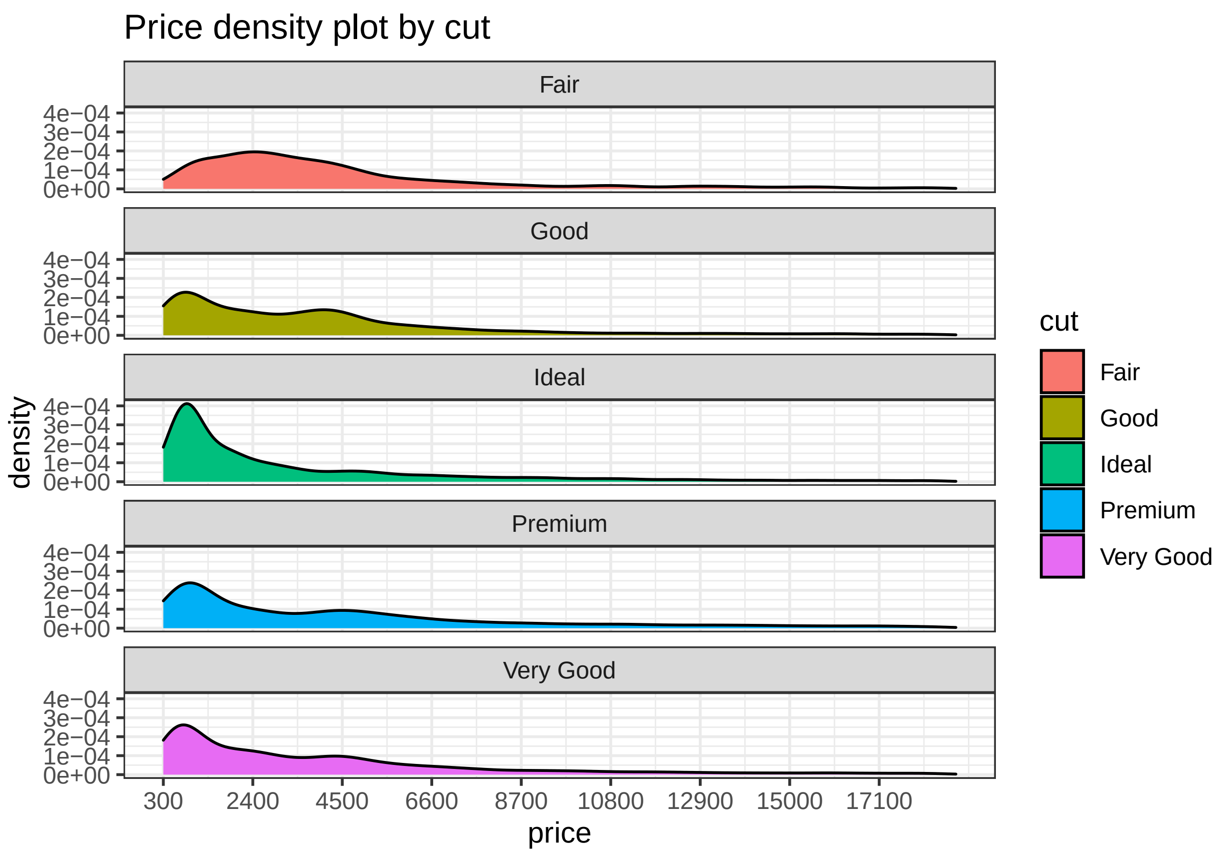


```
ggplot(diamonds) +
  geom_bar(aes(color, fill = color)) +
  labs(title = "Diamonds prevalence by color") +
  scale_fill_brewer(palette = "Greens")
```

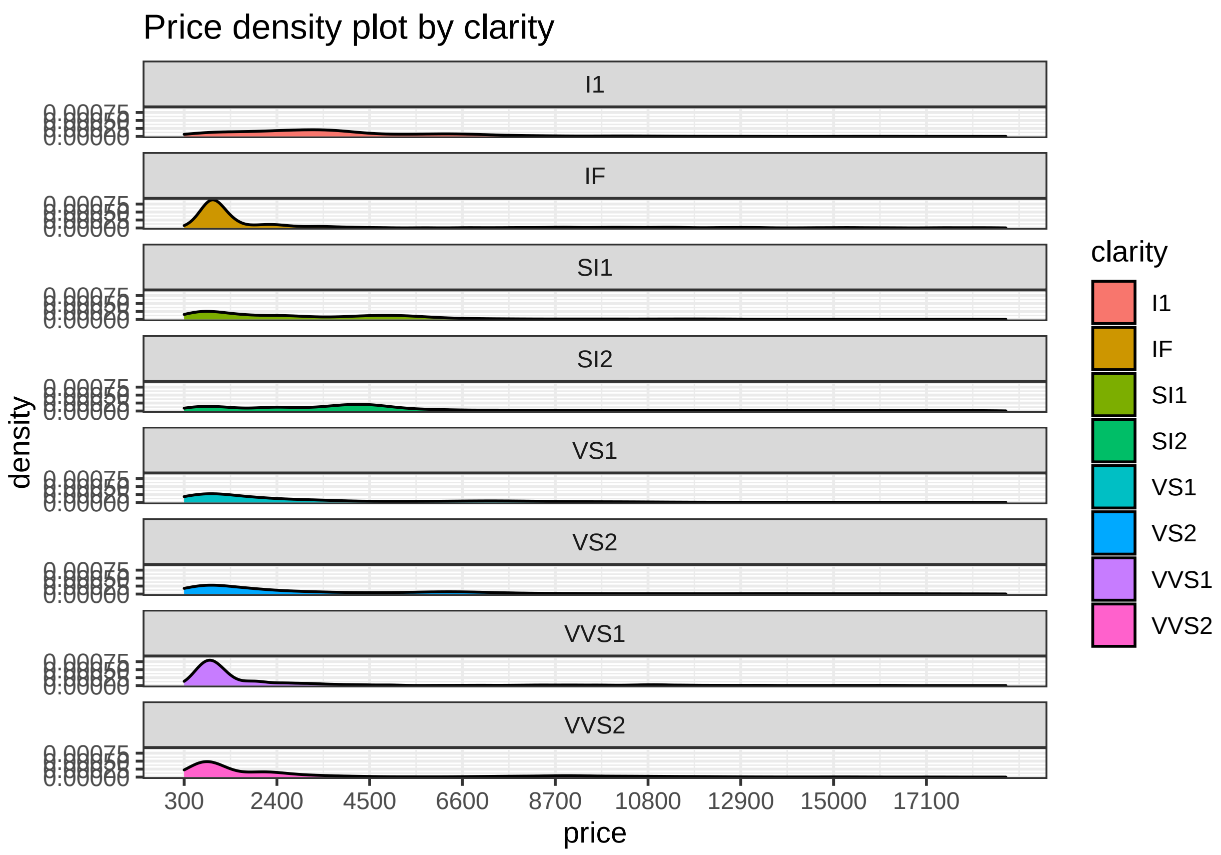


Create price density plot by diamonds `cut`, `clarity` and `color` characteristics.

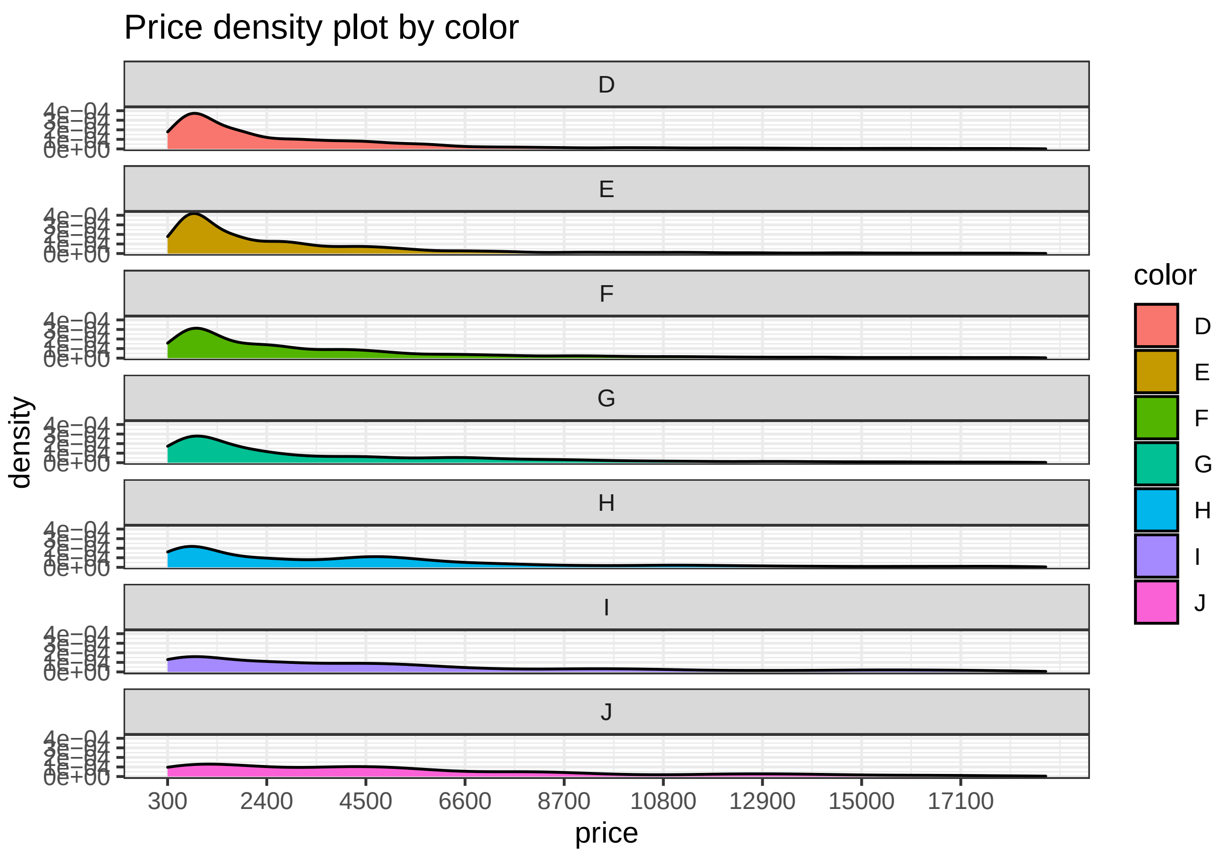
```r
# Price density plot by cut
ggplot(diamonds) +
  geom_density(aes(x = price, fill = cut)) + scale_x_continuous(limit = c(300, 18900),
                                                                breaks = seq(300, 18900, 2100)) +
  facet_wrap(~cut, ncol = 1) +
  theme_bw() +
  ggtitle("Price density plot by cut")
```



```
# Price density plot by clarity
ggplot(diamonds) +
  geom_density(aes(x = price, fill = clarity)) + scale_x_continuous(limit = c(300, 18900),
                                                                    breaks = seq(300, 18900, 2100)) +
  facet_wrap(~clarity, ncol = 1) +
  theme_bw() +
  ggtitle("Price density plot by clarity")
```

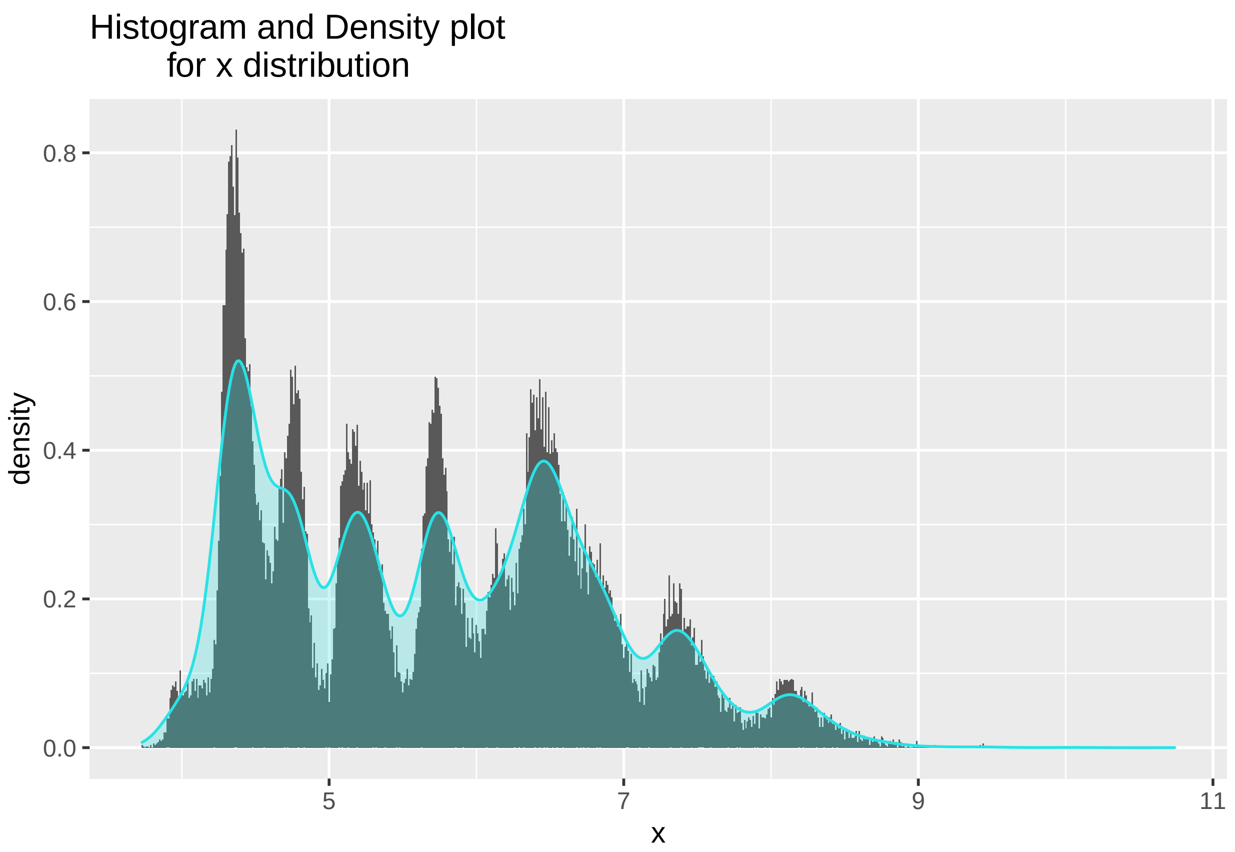


```
# Price density plot by color
ggplot(diamonds) +
  geom_density(aes(x = price, fill = color)) + scale_x_continuous(limit = c(300, 18900),
                                                                    breaks = seq(300, 18900, 2100)) +
  facet_wrap(~color, ncol = 1) +
  theme_bw() +
  ggtitle("Price density plot by color")
```

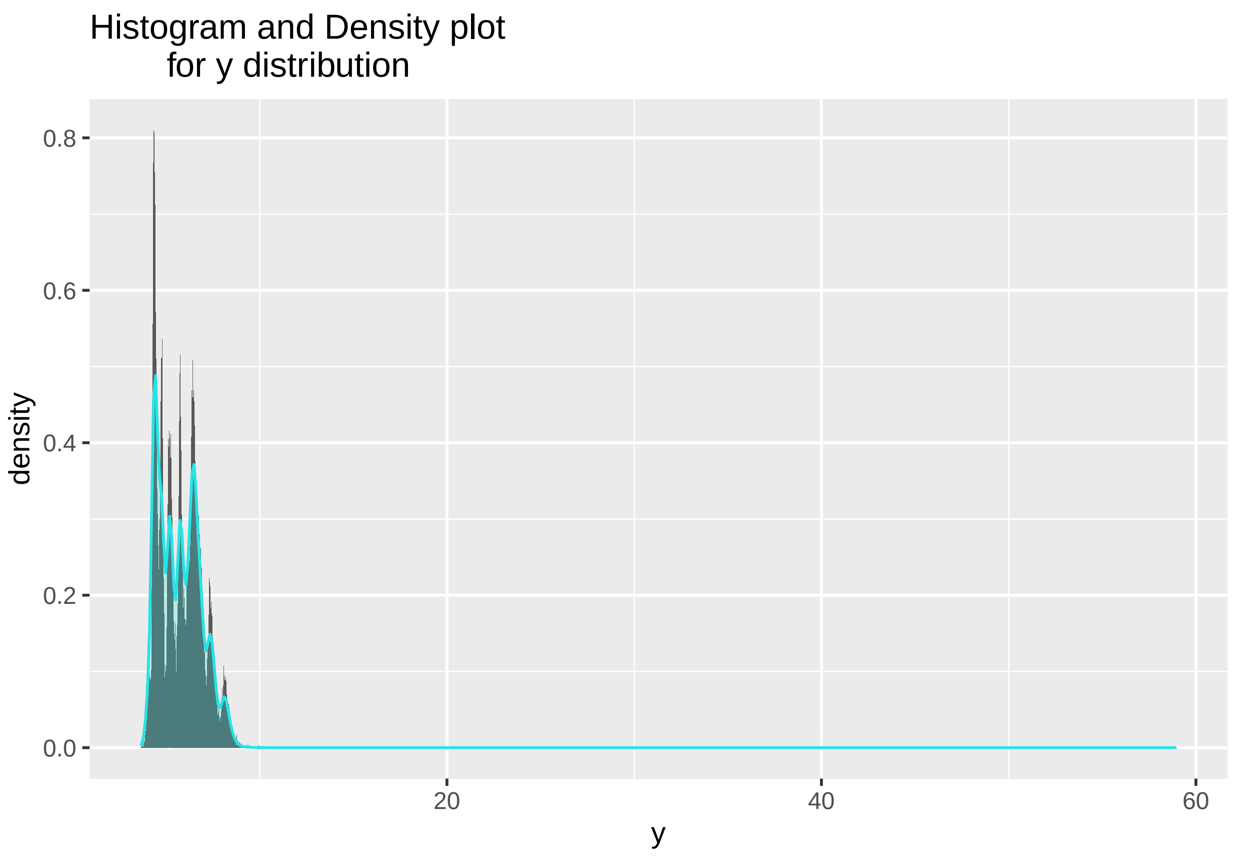


Let's also visualize continuous values of the data using histogram and density plot.

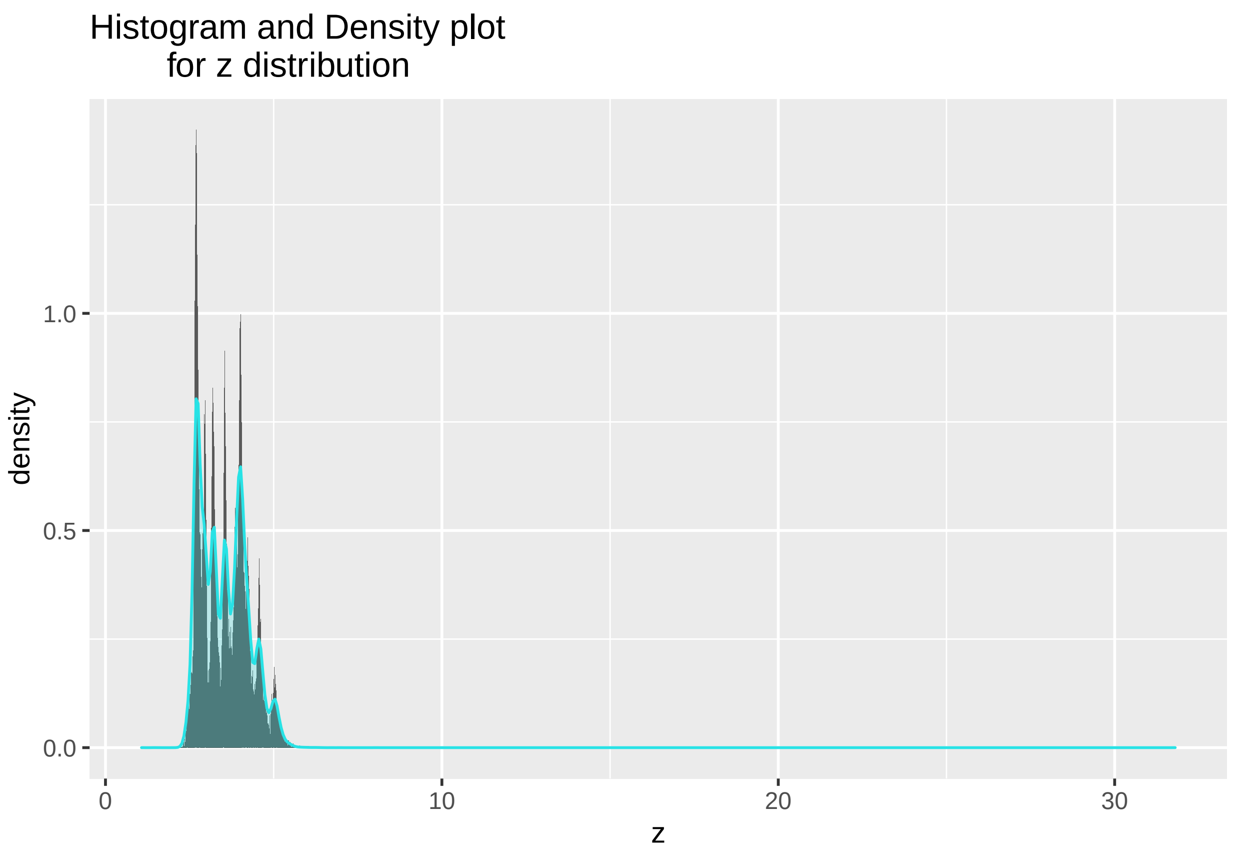
```
# Histogram and density for `x` distribution
ggplot(diamonds) +
  geom_histogram(mapping = aes(x = x, y = ..density..), binwidth = 0.01) +
  geom_density(aes(x = x, y = ..density..),
               lwd = 0.5, color = 5, fill = 5, alpha = 0.25) +
  ggtitle("Histogram and Density plot
        for x distribution")
```



```
# Histogram and density for `y` distribution
ggplot(diamonds) +
  geom_histogram(mapping = aes(x = y, y = ..density..), binwidth = 0.01) +
  geom_density(aes(x = y, y = ..density..),
             lwd = 0.5, color = 5, fill = 5, alpha = 0.25) +
  ggtitle("Histogram and Density plot
        for y distribution")
```



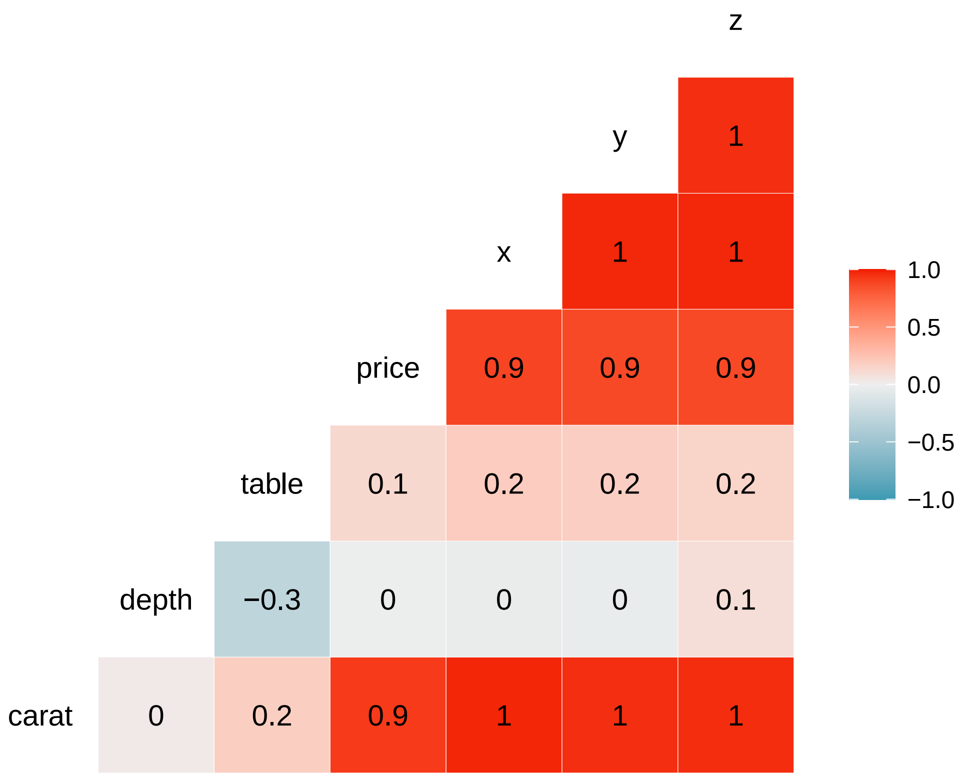
```r
# Histogram and density for `z` distribution
ggplot(diamonds) +
  geom_histogram(mapping = aes(x = z, y = ..density..), binwidth = 0.01) +
  geom_density(aes(x = z, y = ..density..),
               lwd = 0.5, color = 5, fill = 5, alpha = 0.25) +
  ggtitle("Histogram and Density plot
        for z distribution")
```



Afterwards, before creating a machine learning model for price prediction show the correlation matrix, to see which variables are the most correlated with price.

```
# Show correlation matrix to see interconnection between continuous variables
ggcorr(diamonds,
       label = TRUE,
       label_alpha = FALSE)
```

```
## Warning in ggcorr(diamonds, label = TRUE, label_alpha = FALSE): data in
## column(s) 'cut', 'color', 'clarity' are not numeric and were ignored
```



We see a warning because `cut`, `color` and `clarity` are not numeric. But we understand, that these variables are the most important in price prediction. Price is highly correlated with `carat` and diamonds dimensions, such as `x`, `y` and `z`.

DATA MODELING AND RESULTS

Preparing Data for Machine Learning

The data is already visualized and we can see a picture, by virtue of which the further actions can be realized.

Let's change the diamonds `price` and `carats` into ordinal factors.

Convert the categorical variables `cut`, `color` and `clarity` to binary, using `varhandle` package. Assign each of these categories a variable name: `binary_cut`, `binary_clarity`, `binary_color`.

```
# Cutting the diamond price and carats into ordinal factors
diamonds$fprice <- as.numeric(cut(diamonds$price,
                                  seq(from = 0, to = 50000, by = 4000)))

# Convert factored price into ordinal factor
diamonds$fprice <- ordered(diamonds$fprice)

# Diamond's carat is a cut by range 0.5
# `fcarat` is a factored carat
diamonds$fcarat <- as.numeric(cut(diamonds$carat,
                                  seq(from = 0, to = 6, by = 0.1)))

# Convert factored carat into ordinal factor
diamonds$fcarat <- ordered(diamonds$fcarat)
```

```
# Convert categorical variables `cut`, `clarity`, `color` to binary
binary_cut <- to.dummy(diamonds$cut, "cut")

binary_clarity <- to.dummy(diamonds$clarity, "clarity")

binary_color <- to.dummy(diamonds$color, "color")
```

Each categorical quality factor was divided by grade. For instance, if `cut` belongs to the grade `Good` and has the variable is assigned to the `cut.Good`, it has a number `0`(`no`) or `1`(`yes`). This is how binary data looks like:

```
# Check converted data
head(binary_cut)
```

```
##      cut.Fair cut.Good cut.Ideal cut.Premium cut.Very_Good
## [1,]        0        0         1           0             0
## [2,]        0        0         0           1             0
## [3,]        0        1         0           0             0
## [4,]        0        0         0           1             0
## [5,]        0        1         0           0             0
## [6,]        0        0         0           0             1
```

```
head(binary_clarity)
```

```
##      clarity.I1 clarity.IF clarity.SI1 clarity.SI2 clarity.VS1 clarity.VS2
## [1,]          0          0           0           1           0           0
## [2,]          0          0           1           0           0           0
## [3,]          0          0           0           0           1           0
## [4,]          0          0           0           0           0           1
## [5,]          0          0           0           1           0           0
## [6,]          0          0           0           0           0           0
##      clarity.VVS1 clarity.VVS2
## [1,]            0            0
## [2,]            0            0
## [3,]            0            0
## [4,]            0            0
## [5,]            0            0
## [6,]            0            1
```

```
head(binary_color)
```

```
##      color.D color.E color.F color.G color.H color.I color.J
## [1,]       0       1       0       0       0       0       0
## [2,]       0       1       0       0       0       0       0
## [3,]       0       1       0       0       0       0       0
## [4,]       0       0       0       0       0       1       0
## [5,]       0       0       0       0       0       0       1
## [6,]       0       0       0       0       0       0       1
```

All of these actions were performed not only to simplify work with categorical data, but also to speed up data processing for machine learning models and neural networks.

The next step is to assign a new variable of the previous Diamonds dataframe. We will call it `diamonds2`, the second version of the original `diamonds` dataset. In order to implement the intended steps, we need to add newly created columns, and exclude not necessarily variables in future predictions.

```
# Create new `diamonds2` dataframe
# Add newly created columns
```

```
diamonds2 <- cbind(diamonds, binary_cut, binary_clarity, binary_color)

# Exclude from the new dataset `cut`,`color`,`clarity`, `depth`, `table` and
# `x`, `y`, `z` columns
diamonds2 <- subset(diamonds2, select = -c(cut, color, clarity, depth, table,
                                           x, y, z))
```

The updated Diamonds dataframe has a completely different appearance:

```
head(diamonds2)
```

```
##   carat price fprice fcarat cut.Fair cut.Good cut.Ideal cut.Premium
## 1  0.23   326      1      3        0        0         1           0
## 2  0.21   326      1      3        0        0         0           1
## 3  0.23   327      1      3        0        1         0           0
## 4  0.29   334      1      3        0        0         0           1
## 5  0.31   335      1      4        0        1         0           0
## 6  0.24   336      1      3        0        0         0           0
##   cut.Very_Good clarity.I1 clarity.IF clarity.SI1 clarity.SI2 clarity.VS1
## 1             0          0          0           0           1           0
## 2             0          0          0           1           0           0
## 3             0          0          0           0           0           1
## 4             0          0          0           0           0           0
## 5             0          0          0           0           1           0
## 6             1          0          0           0           0           0
##   clarity.VS2 clarity.VVS1 clarity.VVS2 color.D color.E color.F color.G color.H
## 1           0            0            0       0       1       0       0       0
## 2           0            0            0       0       1       0       0       0
## 3           0            0            0       0       1       0       0       0
## 4           1            0            0       0       0       0       0       0
## 5           0            0            0       0       0       0       0       0
## 6           0            0            1       0       0       0       0       0
##   color.I color.J
## 1       0       0
## 2       0       0
## 3       0       0
## 4       1       0
## 5       0       1
## 6       0       1
```

The data has too many observations. Let's minimize the number of observations to `10,000`. Create new dataframe `diamonds3`, which will be used in machine learning.

```
# Assign the number of observations to `10000` to make the dataset smaller
num_obs <- 10000

# Sample data from the subset based on the number of observations
sample_data <- diamonds2[sample(1:nrow(diamonds2), num_obs,
                               replace=FALSE),]

# Create new dataframe with sampled data
diamonds3 <- sample_data
```

New `diamonds3` dataframe:

```r
head(diamonds3)
```

```
##       carat price fprice fcarat cut.Fair cut.Good cut.Ideal cut.Premium
## 2529   1.11  3205      1     12        0        0         0           1
## 34939  0.27   377      1      3        0        1         0           0
## 50769  0.71  2300      1      8        0        0         0           1
## 50589  0.70  2282      1      7        0        1         0           0
## 39192  0.36  1064      1      4        0        0         1           0
## 33332  0.41   827      1      5        0        0         0           0
##       cut.Very_Good clarity.I1 clarity.IF clarity.SI1 clarity.SI2 clarity.VS1
## 2529              0          1          0           0           0           0
## 34939             0          0          0           0           0           1
## 50769             0          0          0           0           0           0
## 50589             0          0          0           0           0           0
## 39192             0          0          0           0           0           1
## 33332             1          0          0           0           0           0
##       clarity.VS2 clarity.VVS1 clarity.VVS2 color.D color.E color.F color.G
## 2529            0            0            0       0       0       0       0
## 34939           0            0            0       0       1       0       0
## 50769           1            0            0       0       0       0       0
## 50589           1            0            0       0       0       0       1
## 39192           0            0            0       1       0       0       0
## 33332           1            0            0       0       0       0       1
##       color.H color.I color.J
## 2529        1       0       0
## 34939       0       0       0
## 50769       0       1       0
## 50589       0       0       0
## 39192       0       0       0
## 33332       0       0       0
```

Machine Learning Approaches Explanation

In the next section will be implemented two non-trivial (in comparison with *linear* and *logistic* regression) machine learning algorithms: *Decision Tree* and *Random Forest*. In the prediction part will be used *Confusion Matrix*, to compute the *accuracy* of our models.

Here is a brief explanation of these methods.

Decision Tree

This algorithm is used for predictions in the form of a `tree structure` where each internal node denotes a `test` on an `attribute`, each `branch` represents an outcome of the `test`, and each `leaf node` (terminal node) holds a class label. To construct a `decision tree` the algorithm uses *Entropy* and *Information Gain*, firstly invented by J. R. Quinlan[3]

In few words, *Entropy* helps in partitioning of the data into subsets that contain instances with similar values.

Information Gain is based on the decrease in entropy after a dataset is split on an attribute. Constructing a `decision tree` is all about finding attribute that returns the highest information gain (i.e., the most similar branches).

`Entropy` formula:

[3]https://link.springer.com/content/pdf/10.1007%2FBF00116251.pdf

$$E(A) = \sum_{i=I}^{V} \frac{p_i + n_i}{p + n} I(p_i, n_i)$$

where `A(root)` is the information required for a tree;

$$I(p_i, n_i)$$

is the expected information required for a subtree;

And this formula is the `information gained` by branching on `A`:

$$gain(A) = I(p, n) - E(A)$$

Random Forest

Random forest algorithm is made up of multiple decision trees. It have three main hyperparameters, which need to be set before training. These include node size, the number of trees, and the number of features sampled.

From there, the random forest classifier can be used to solve for regression or classification problems.

The `Random Forest` randomly splits the data and creates many trees, or a forest.

Confusion Matrix

The confusion matrix is a table which calibrates the output of a model and examining all possible outcomes of the predictions. It uses cross tabulation of the predicted values with the actual values.

	Actual Positive	Actual Negative
Predicted Positive	True Positive (TP)	False Positive (FP)
Predicted Negative	False Negative (FN)	True Negative (TN)

The meaning of each value in the table:

`True Positive (TP):` predicted *positive* for an actual *positive* value.

`True Negative (TN):` predicted *negative* for an actual *negative* value.

`False Positive (FN):` predicted *positive* for an actual *negative* value.

`False Negative (FN):` predicted *negative* for an actual *positive* value.

Confusion matrix can be useful to compute the following statistical metrics:

`Accuracy:` the proportion of correct predictions for both *positive* and *negative* outcomes, i.e. the ability to correctly predict a *positive* and *negative.*

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

`Sensitivity:` the proportion of *positive* values when they are actually *positive*, i.e. the ability to predict *positive* values.

$$Sensitivity = \frac{TP}{TP + FN}$$

`Specificity` specificity is the proportion of *negative* values when they are actually *negative*, i.e. the ability to predict *negative* values.

$$Specificity = \frac{TN}{TN + FP}$$

Data Partitioning for Machine Learning

Split `diamonds3` dataset into two subsets: one for training and another one for testing. Train set should be divided by 80%, and the test set (validation) should be split by 20% respectively.

Show a confusion matrix in the end of predictions for the validation set and check the `accuracy` of each algorithm used for predictions.

```
## Create `train` and `test` sets for `diamonds3` dataframe

set.seed(2021, sample.kind = "Rounding") # using R 3.5 or earlier, use `set.seed(1)`
```

```
## Warning in set.seed(2021, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```
# Split data into train and test sets
test_index <- createDataPartition(y = diamonds3$fprice, times = 1, p = 0.8, list = FALSE)
test_set <- diamonds3[test_index, ]
train_set <- diamonds3[-test_index,]
```

Model 1: Decision Tree

Fit the model for `Decision Tree` algorithm. Write predictions for the `train` and `test` set. Show the confusion matrix of the `validation` set.

```
# Fit the ml model
dtree_fit <- train(fprice ~., method = "rpart", data = train_set)
```

```
## Warning in (function (kind = NULL, normal.kind = NULL, sample.kind = NULL) :
## non-uniform 'Rounding' sampler used
```

```
# Predictions on the train set
pred_train <- predict(dtree_fit, train_set)

# Predictions on the test set
pred_test <- predict(dtree_fit, test_set)

# Write the confusion matrix to see the accuracy,
# sensitivity and specificity of the predicted test set
print(confusionMatrix(pred_test, test_set$fprice))
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction    1    2    3    4    5
##          1 5106    0    0    0    0
##          2    0 1710    0    0    0
##          3    0    0  649  360  176
##          4    0    0    0    0    0
##          5    0    0    0    0    0
##
## Overall Statistics
```

```
##
##                Accuracy : 0.933
##                  95% CI : (0.9273, 0.9384)
##     No Information Rate : 0.6382
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8748
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            1.0000   1.0000  1.00000  0.00000    0.000
## Specificity            1.0000   1.0000  0.92709  1.00000    1.000
## Pos Pred Value         1.0000   1.0000  0.54768      NaN      NaN
## Neg Pred Value         1.0000   1.0000  1.00000  0.95501    0.978
## Prevalence             0.6382   0.2137  0.08111  0.04499    0.022
## Detection Rate         0.6382   0.2137  0.08111  0.00000    0.000
## Detection Prevalence   0.6382   0.2137  0.14811  0.00000    0.000
## Balanced Accuracy      1.0000   1.0000  0.96355  0.50000    0.500
```

Accuracy of the `Decision Tree` algorithm predictions is 93%

Model 2: Random Forest

Fit the model for `Decision Tree` algorithm. Write predictions for the `train` and `test` set. Set the number of `trees` to 10. Show the confusion matrix of the `validation` set.

```r
# Fit the ml model
rforest_fit <- randomForest(fprice ~. , data = train_set,
                       importance = TRUE, ntrees = 10)

# Predictions on the train set
pred_train2 <- predict(rforest_fit, train_set)

# Predictions on the test set
pred_test2 <- predict(rforest_fit, test_set)

# Write the confusion matrix to see the accuracy,
# sensitivity and specificity of the predicted test set
print(confusionMatrix(pred_test2, test_set$fprice))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    1    2    3    4    5
##          1 5106    4    0    0    0
##          2    0 1706    3    0    0
##          3    0    0  637   27    6
##          4    0    0    9  330   26
##          5    0    0    0    3  144
##
## Overall Statistics
##
##                Accuracy : 0.9903
##                  95% CI : (0.9878, 0.9923)
##     No Information Rate : 0.6382
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9819
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            1.0000   0.9977  0.98151  0.91667  0.81818
## Specificity            0.9986   0.9995  0.99551  0.99542  0.99962
## Pos Pred Value         0.9992   0.9982  0.95075  0.90411  0.97959
## Neg Pred Value         1.0000   0.9994  0.99836  0.99607  0.99593
## Prevalence             0.6382   0.2137  0.08111  0.04499  0.02200
## Detection Rate         0.6382   0.2132  0.07962  0.04124  0.01800
## Detection Prevalence   0.6387   0.2136  0.08374  0.04562  0.01837
## Balanced Accuracy      0.9993   0.9986  0.98851  0.95604  0.90890
```

Accuracy of the `Random Forest` algorithm predictions is 99%, which is better then was in `Decision Tree`.

Addendum*: Artificial Neural Network

`Artificial Neural Network`[4] is a mean of doing machine learning.

The basic units to understand the neural network (also called as `Perceptron`[5]):

`Neuron` or `Node` takes inputs, and produces the outputs;

`Weight` is a connection between *neurons* that carries a value;

`Layer` connects directly to an input variable and contributes to an output variable and it has `3 types`:

`Input Layer` or `Visible Layer` consists of the input variables;

`Hidden Layers` are the layers of nodes between the input and output layers. There may be one or more of these layers;

`Output Layer` is a layer of nodes that produce the output variables.

`Bias (Threshold)` just like the weight, carries a value. (Every neuron that is not on the input layer has a `Bias` attached to it.)

The neural network used in this report is `feed-forward`, which means that the data moves through layers of nodes in only one direction.

To start building the nn, it is important to normalize data. This time we will use `Min-Max Normalization`, which corresponds to the following formula:

$$x' = a + \frac{x - min(x))(b - a)}{max(x) - min(x)}$$

`Min-Max Normalization` transforms x to x' by converting each value of features to a range between 0 and 1, and this is also known as (0–1) Normalization. If the data has negative values the range would have been between -1 and 1.

Create a formula to normalize data from `diamonds3` dataset.

[4]https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414

[5]https://cs.stanford.edu/people/eroberts/courses/soco/projects/neural-networks/Neuron/index.html

```
# Before creating nn we should normalize our data using `min-max normalization`
normalize <- function(x){
  (x- min(x)) /(max(x)-min(x))
}
```

Assign normalized data to `diamonds4` variable and select only numeric features. After that, create new Diamonds dataframe `diamonds5` to be able work with data, specially made for the neural network.

```
diamonds4 <-  diamonds3 %>% select_if(is.numeric)

# Normalize diamonds data from `diamonds4` dataset
# Set the `diamonds5` name to the normalized dataframe
diamonds5 <- as.data.frame(lapply(diamonds4, normalize))

# Check the first six rows of the updated data
head(diamonds5)
```

```
##        carat       price cut.Fair cut.Good cut.Ideal cut.Premium cut.Very_Good
## 1 0.21162791 0.155882831        0        0         0           1             0
## 2 0.01627907 0.002761384        0        1         0           0             0
## 3 0.11860465 0.106881802        0        0         0           1             0
## 4 0.11627907 0.105907196        0        1         0           0             0
## 5 0.03720930 0.039958850        0        0         1           0             0
## 6 0.04883721 0.027126536        0        0         0           0             1
##   clarity.I1 clarity.IF clarity.SI1 clarity.SI2 clarity.VS1 clarity.VS2
## 1          1          0           0           0           0           0
## 2          0          0           0           0           1           0
## 3          0          0           0           0           0           1
## 4          0          0           0           0           0           1
## 5          0          0           0           0           1           0
## 6          0          0           0           0           0           1
##   clarity.VVS1 clarity.VVS2 color.D color.E color.F color.G color.H color.I
## 1            0            0       0       0       0       0       1       0
## 2            0            0       0       1       0       0       0       0
## 3            0            0       0       0       0       0       0       1
## 4            0            0       0       0       0       1       0       0
## 5            0            0       1       0       0       0       0       0
## 6            0            0       0       0       0       1       0       0
##   color.J
## 1       0
## 2       0
## 3       0
## 4       0
## 5       0
## 6       0
```

Method 3: Neural Network

The Diamonds dataset is prepared for data partitioning. Split data as in machine learning section into two subsets: one for training (80%) and one for validation (20%).

```
# Data partitioning for neural networks
test_index_n <- sample(nrow(diamonds5), 0.8 * nrow(diamonds5))
train_set_n <- diamonds5[test_index_n, ]
test_set_n <- diamonds5[-test_index_n, ]
```

The number of observations in the `train` set is 8000, and the number of observations in `test` set is 2000.

```
nrow(train_set_n)
```

```
## [1] 8000
```

```
nrow(test_set_n)
```

```
## [1] 2000
```

Finally, create neural network models using `neuralnet`[6] package.

Assign a variable `NN` to the `train` set and perform the same operation for another variable `nn2` to the `validation` set. The `target` variable which we want to predict is `price`, that is why it goes as the first command while calling `neuralnet` function. Our predictions would be of the all diamonds attributes which are included in the `diamonds5` dataset.

To improve the accuracy of the `ANN` predictions at the first time will be generated 5 hidden layers, and after that our calculations will go again through 3 hidden layers. Assign the linear output to `TRUE` value.

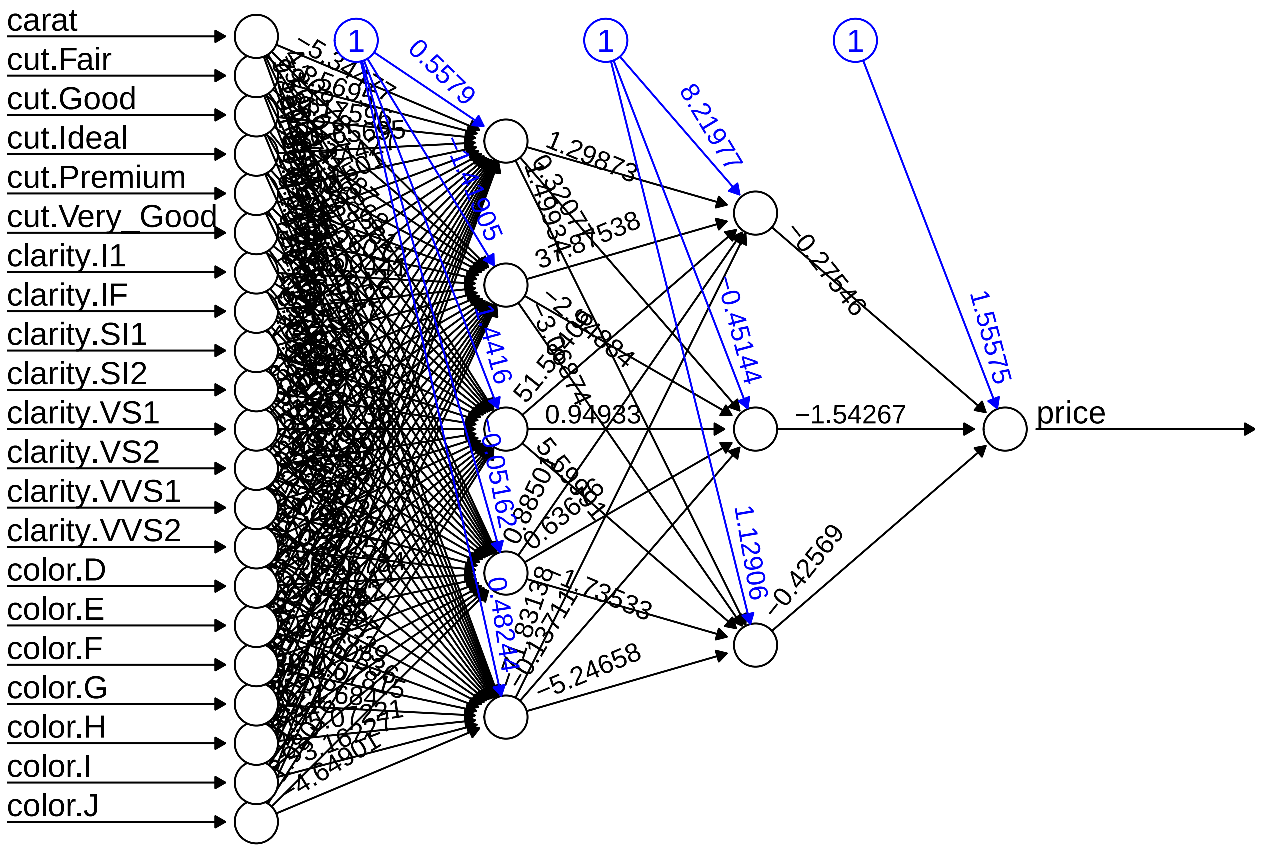
```
# Assign a variable `nn` for the `train_set_n`
# Please, wait... The process may take several minutes
# It depends of the computer resources
nn <- neuralnet(price~carat+cut.Fair+cut.Good+cut.Ideal+cut.Premium+cut.Very_Good+
                  clarity.I1+clarity.IF+clarity.SI1+clarity.SI2+clarity.VS1+
                  clarity.VS2+clarity.VVS1+clarity.VVS2+
                  color.D+color.E+color.F+color.G+
                  color.H+color.I+color.J,
                data = train_set_n, hidden = c(5, 3),
                linear.output = TRUE)

# Assign a variable `nn2` for the `test_set_n`
nn2 <- neuralnet(price~carat+cut.Fair+cut.Good+cut.Ideal+cut.Premium+cut.Very_Good+
                   clarity.I1+clarity.IF+clarity.SI1+clarity.SI2+clarity.VS1+
                   clarity.VS2+clarity.VVS1+clarity.VVS2+
                   color.D+color.E+color.F+color.G+
                   color.H+color.I+color.J,
                 data = test_set_n, hidden = c(5, 3),
                 linear.output = TRUE)
```

Show the appearance of `NN` for the `validation` set.

```
plot(nn2, rep = "best")
```

[6] https://www.rdocumentation.org/packages/neuralnet/versions/1.44.2/topics/neuralnet



Predict the outputs for `training` and `testing` sets.

```
# Predictions for training set
predict_train <- compute(nn, train_set_n)

# Predictions for testing set
predict_test <- compute(nn2, test_set_n)
```

Model Validation

Show actual and predicted results for the validation set. Check the first rows for this set.

```
# Actual and prediction results for the validation set
results_test <- data.frame(actual = test_set_n$price,
                           prediction = predict_test$net.result)
head(results_test)
```

```
##          actual prediction
## 5  0.039958850 0.02509675
## 12 0.009637771 0.01798870
## 18 0.183496670 0.16259452
## 20 0.048297147 0.03973124
## 21 0.093562185 0.09875081
## 34 0.289403866 0.26033003
```

```
# Test the accuracy of our model
predicted <- results_test$prediction * abs(diff(range(test_set_n$price))) +
  min(test_set_n$price)
actual <- results_test$actual * abs(diff(range(test_set_n$price))) + min(test_set_n$price)
```

```
comparison <- data.frame(predicted, actual)
deviation <- ((actual - predicted)/actual)
comparison <- data.frame(predicted, actual, deviation)
accuracy <- 1 - abs(mean(deviation))

accuracy
```

```
## [1] 0.9952714
```

Accuracy of price prediction varies depending on the number of runs of the neural network. At the first time it was 99%, the second time it was 95%, the third time it was 97%. The decision was to consider its accuracy by the first time.

Algorithms performed in comparison with each other:

Model	Decision Tree	Random Forest	Neural Network
Accuracy	93%	99%	99%

CONCLUSION

During this exploratory data analysis with Diamonds dataset were performed several steps.

At the very beginning were uploaded Diamonds data and discovered its structure. We understood how each diamonds attribute correlates with price. Step-by-step were visualized all the interconnections between the diamonds quality factors.

In the middle of data analysis, were briefly explained the basics of machine learning algorithms (`Decision Tree`, `Random Forest`) and other data science approaches for predictions (`Accuracy`, `Confusion Matrix` etc.). As an addendum to the project, were explained and performed data manipulations for building an `Artificial Neural Network`.

In the end, we compared the accuracy of each method for diamonds price prediction.

Limitations

It can be possible to make a statement that all of the chosen methods in diamond price prediction performed satisfying results.

`Decision Tree` algorithm were performed the prediction of the model fast and the achieved accuracy was 93%.

`Random Forest` also gave us a high degree of accuracy: 99%. The model process took the same amount of time for a laptop computations, as well as in the `Decision Tree` algorithm.

`Artificial Neural Network` performed the best prediction at the first time of running the model, more than 99,9%. If we will run it with iterations, we would be able to receive another predictions 95%-98%, but we will always go back to the result we got the first time.

In general, machine learning algorithms can capture decent results as well as neural networks. In comparison, execution speed of calculations differs. `ANN` take more time for processing, but it also depends of the quantity of the hidden layers and current model. In some data science projects accuracy can be neglected by time speed.

Future work

There are also other neural network models which were not discussed here: Recurrent NN, Deep Feed Forward (DFF), Markov Chain (MC), Neural Turing Machine (NTM) etc.

REFERENCES

Natural Diamonds|Diamonds Stone https://www.gia.edu/diamond

5 Factors Of A Diamond Cut Quality Comparison https://www.brilliance.com/diamonds/factors-diamond-cut-quality-comparison

What is the Ideal Depth of a Diamond? https://blog.brilliance.com/diamonds/what-is-the-ideal-depth-of-a-diamond

Diamond Color Chart https://www.lumeradiamonds.com/diamond-education/diamond-color

Data Normalization With R https://medium.com/swlh/data-normalisation-with-r-6ef1d1947970

The Mathematics of Neural Networks https://medium.com/coinmonks/the-mathematics-of-neural-network-60a112dd3e05

What are neural networks? https://www.ibm.com/cloud/learn/neural-networks