StudyR1

PX

3/6/2020

Reading Data

```
# comma separated values (CSV)

# use read.table

theURL <-"http://www.jaredlander.com/data/Tomato%20First.csv"
tomato <- read.table(file = theURL,header = TRUE, sep = ",")
head(tomato)
```

```
##   Round             Tomato Price      Source Sweet Acid Color Texture Overall
## 1     1         Simpson SM  3.99 Whole Foods   2.8  2.8   3.7     3.4     3.4
## 2     1  Tuttorosso (blue)  2.99     Pioneer   3.3  2.8   3.4     3.0     2.9
## 3     1 Tuttorosso (green)  0.99     Pioneer   2.8  2.6   3.3     2.8     2.9
## 4     1     La Fede SM DOP  3.99   Shop Rite   2.6  2.8   3.0     2.3     2.8
## 5     2       Cento SM DOP  5.49  D Agostino   3.3  3.1   2.9     2.8     3.1
## 6     2      Cento Organic  4.99  D Agostino   3.2  2.9   2.9     3.1     2.9
##   Avg.of.Totals Total.of.Avg
## 1          16.1         16.1
## 2          15.3         15.3
## 3          14.3         14.3
## 4          13.4         13.4
## 5          14.4         15.2
## 6          15.5         15.1
```

```
# one of the argument in read.table is stringASFactors,
# if we set it as FALSE, then character will not be converted
# into factor
# sometimes it will save a lot computational time
# It also can be used in data.frame argument

# If CSVs are poorly built, read.csv2 or read.delim2, read.delim



# Excel Data
# it will be easy if we convert Excel into CSV file.

# DSN Data source connection

# RData file is very handy to share and adpot

save(tomato, file = "~/Documents/OMSA/ISYE6501/tomato.rdata")
```

```
rm(tomato)

#load the data
load("~/Documents/OMSA/ISYE6501/tomato.rdata")
head(tomato)
```

```
##   Round             Tomato Price      Source Sweet Acid Color Texture Overall
## 1     1         Simpson SM  3.99 Whole Foods   2.8  2.8   3.7     3.4     3.4
## 2     1  Tuttorosso (blue)  2.99     Pioneer   3.3  2.8   3.4     3.0     2.9
## 3     1 Tuttorosso (green)  0.99     Pioneer   2.8  2.6   3.3     2.8     2.9
## 4     1     La Fede SM DOP  3.99   Shop Rite   2.6  2.8   3.0     2.3     2.8
## 5     2       Cento SM DOP  5.49  D Agostino   3.3  3.1   2.9     2.8     3.1
## 6     2      Cento Organic  4.99  D Agostino   3.2  2.9   2.9     3.1     2.9
##   Avg.of.Totals Total.of.Avg
## 1          16.1         16.1
## 2          15.3         15.3
## 3          14.3         14.3
## 4          13.4         13.4
## 5          14.4         15.2
## 6          15.5         15.1
```

```
# create a data.frame

n <- 20
r <- 1:17
w <- data.frame(n,r)
w
```

```
##     n  r
## 1  20  1
## 2  20  2
## 3  20  3
## 4  20  4
## 5  20  5
## 6  20  6
## 7  20  7
## 8  20  8
## 9  20  9
## 10 20 10
## 11 20 11
## 12 20 12
## 13 20 13
## 14 20 14
## 15 20 15
## 16 20 16
## 17 20 17
```

```
save(n,r,w, file = "~/Documents/OMSA/ISYE6501/multiple.rdata")
rm(n,r,w)
load("~/Documents/OMSA/ISYE6501/multiple.rdata")
n
```

```
## [1] 20
```

Statistical Graphics

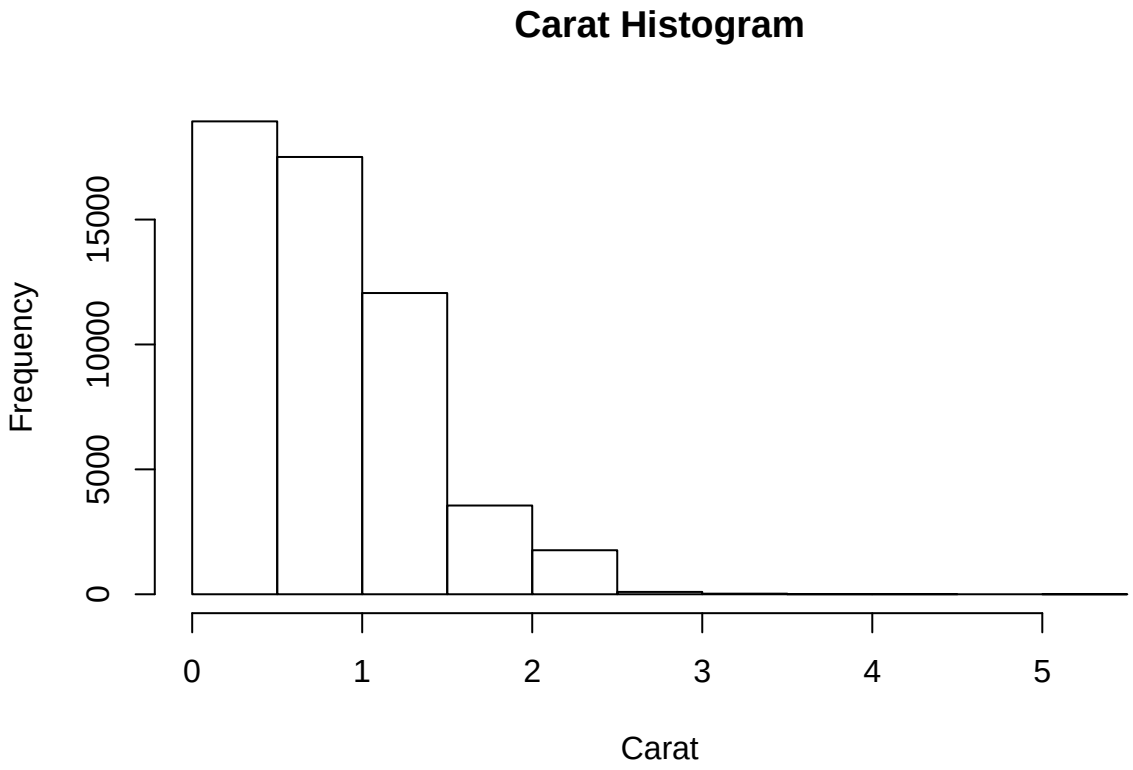
```
# Exploratory Data Analysis - EDA

library(ggplot2)
diamonds
```

```
## # A tibble: 53,940 x 10
##    carat cut       color clarity depth table price     x     y     z
##    <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
##  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 0.290 Premium   I     VS2      62.4    58   334  4.2   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
##  7 0.24  Very Good I     VVS1     62.3    57   336  3.95  3.98  2.47
##  8 0.26  Very Good H     SI1      61.9    55   337  4.07  4.11  2.53
##  9 0.22  Fair      E     VS2      65.1    61   337  3.87  3.78  2.49
## 10 0.23  Very Good H     VS1      59.4    61   338  4     4.05  2.39
## # ... with 53,930 more rows
```

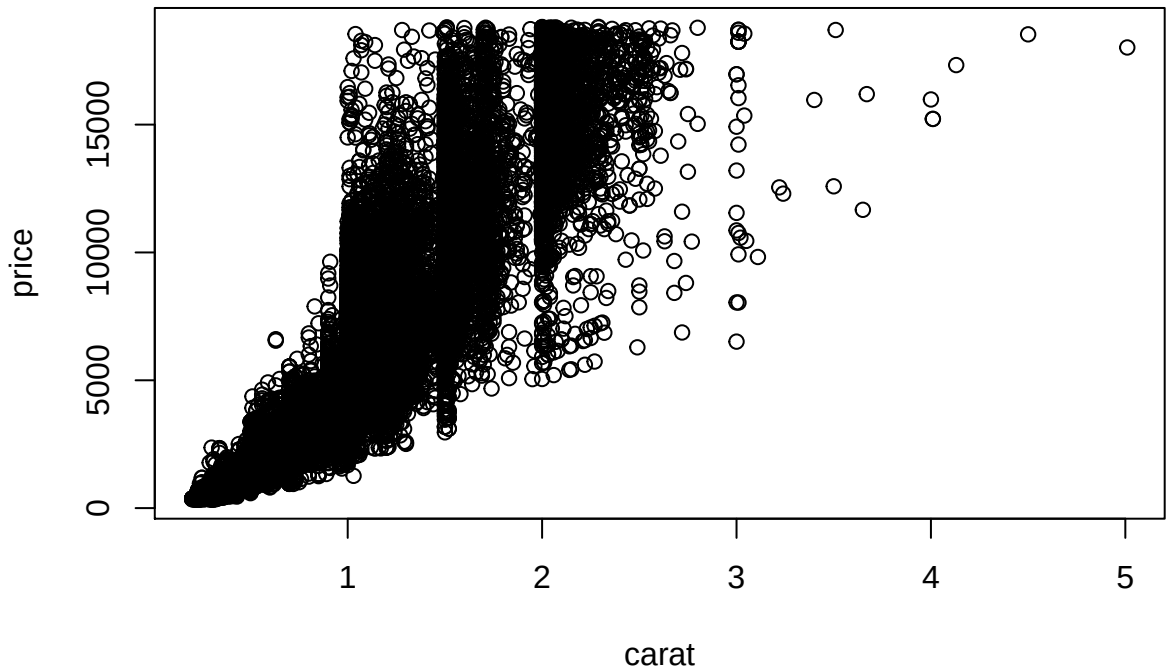
```
head(diamonds)
```

```
## # A tibble: 6 x 10
##   carat cut       color clarity depth table price     x     y     z
##   <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
## 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 0.290 Premium   I     VS2      62.4    58   334  4.2   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
```

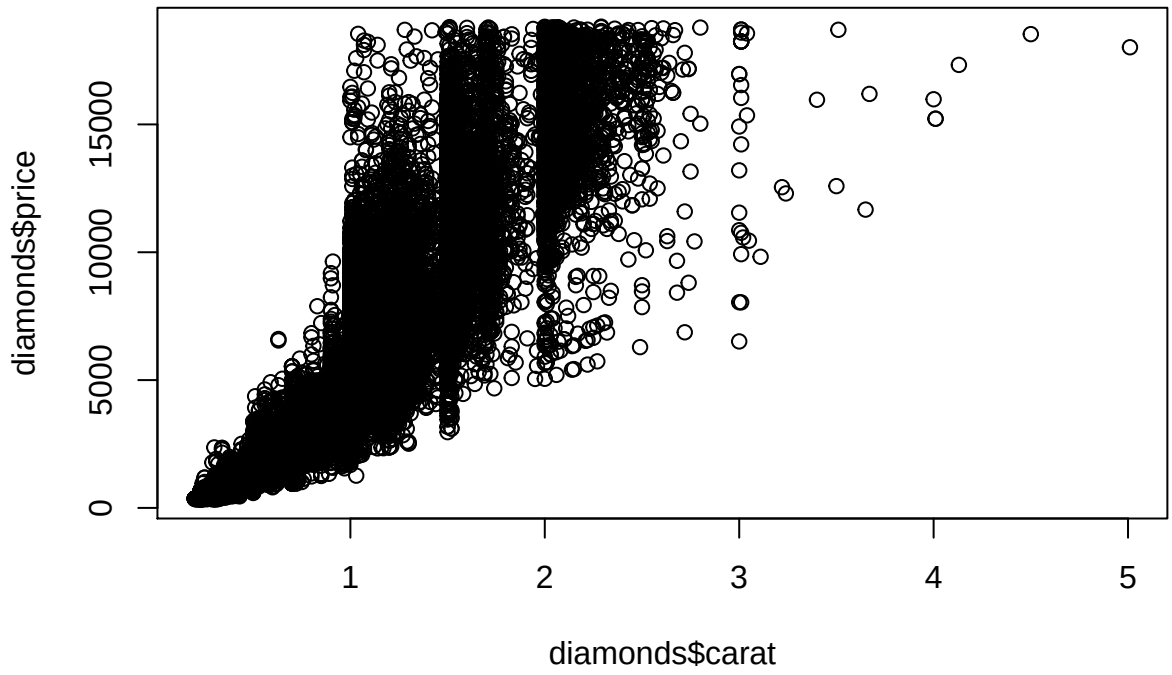
```
# Histogram

hist(diamonds$carat, main = "Carat Histogram",  xlab = "Carat")
```



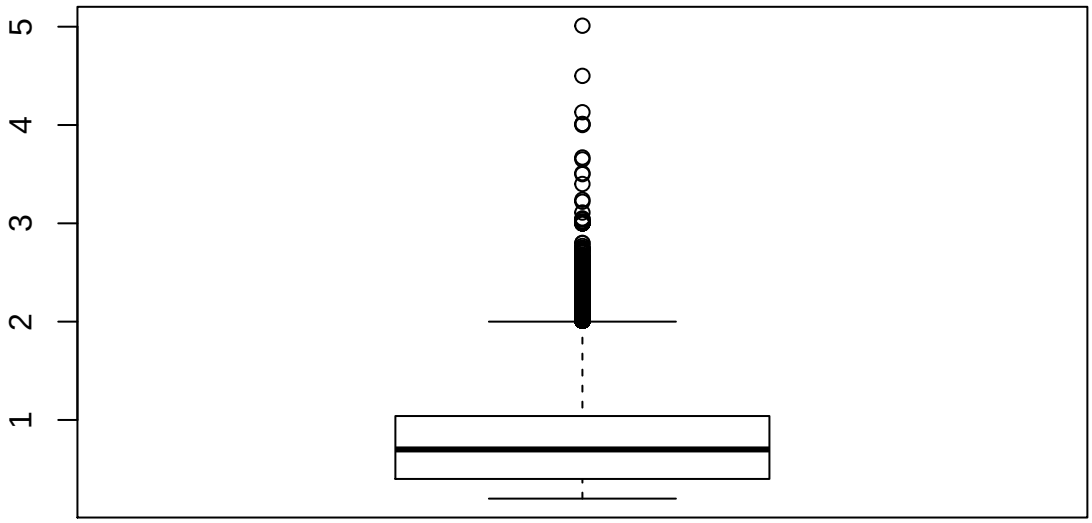

```
# scatter plot
plot(price ~ carat, data = diamonds)
```




```
# ~ separating price and carat, y will be price and x will be carat.

plot(diamonds$carat, diamonds$price)
```

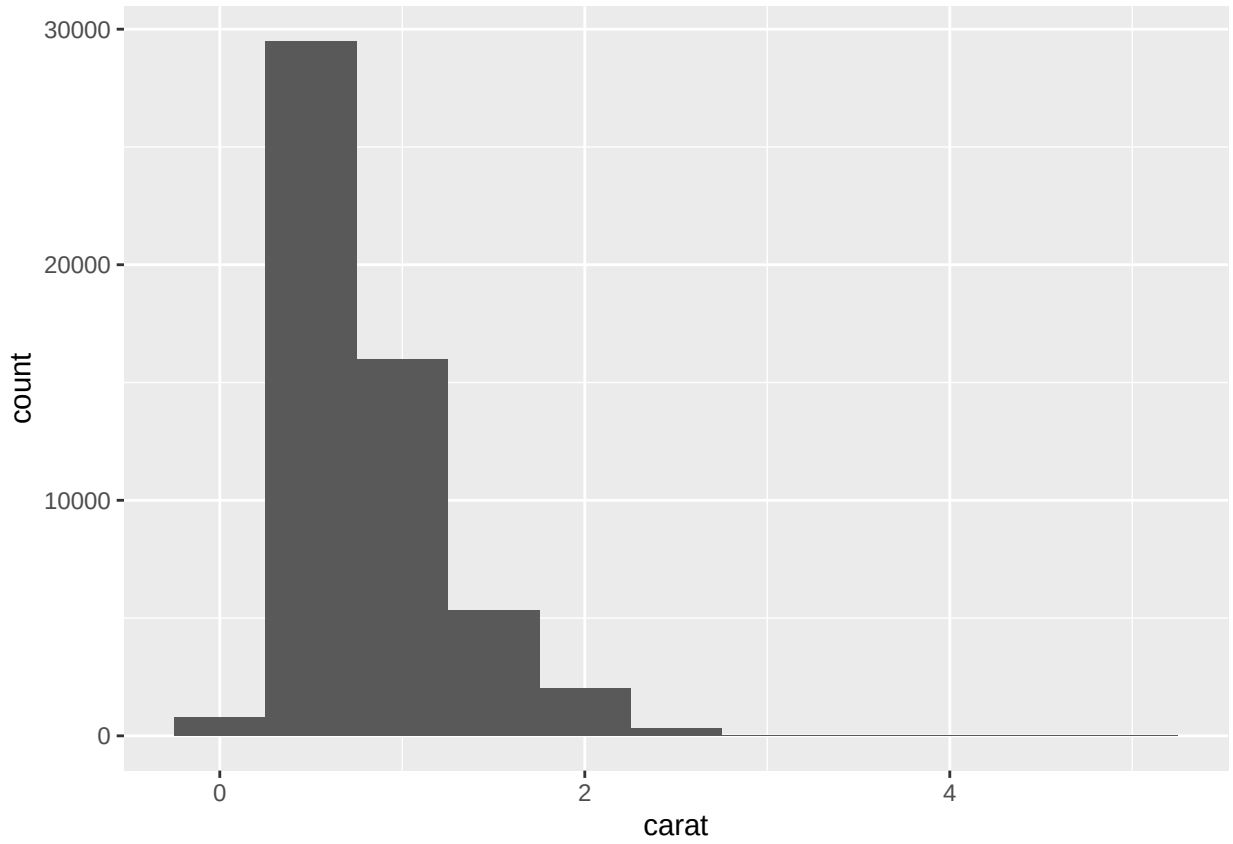



```
#boxplot
boxplot(diamonds$carat)
```

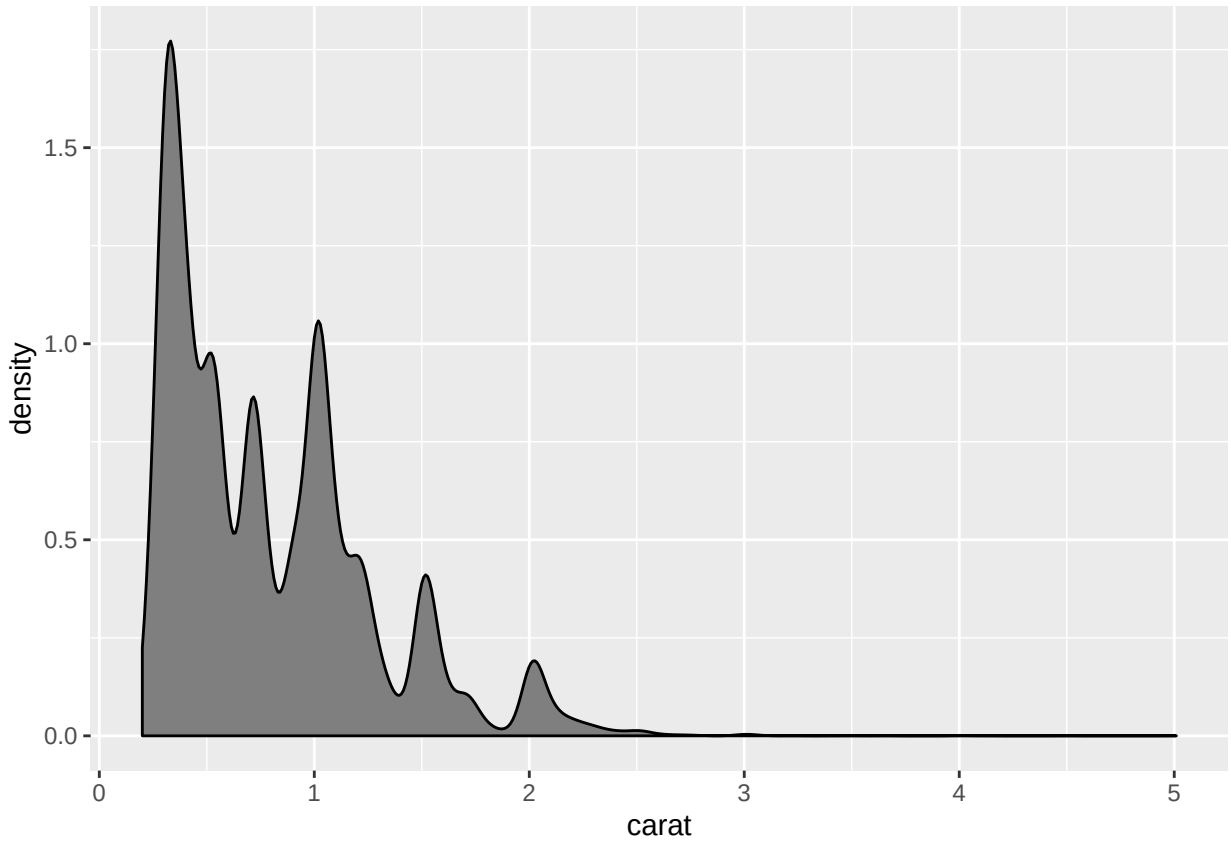


GGPLOT2

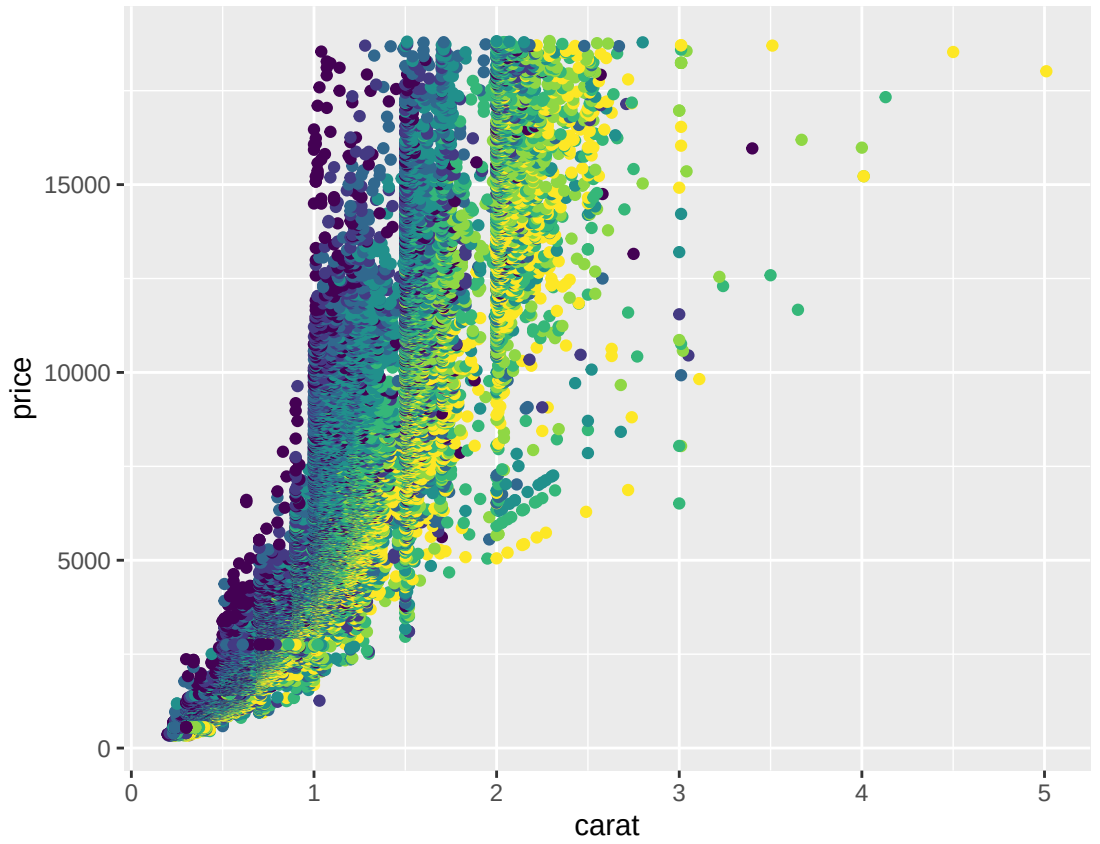
```
ggplot(data = diamonds)+geom_histogram(aes(x=carat),binwidth = 0.5)
```



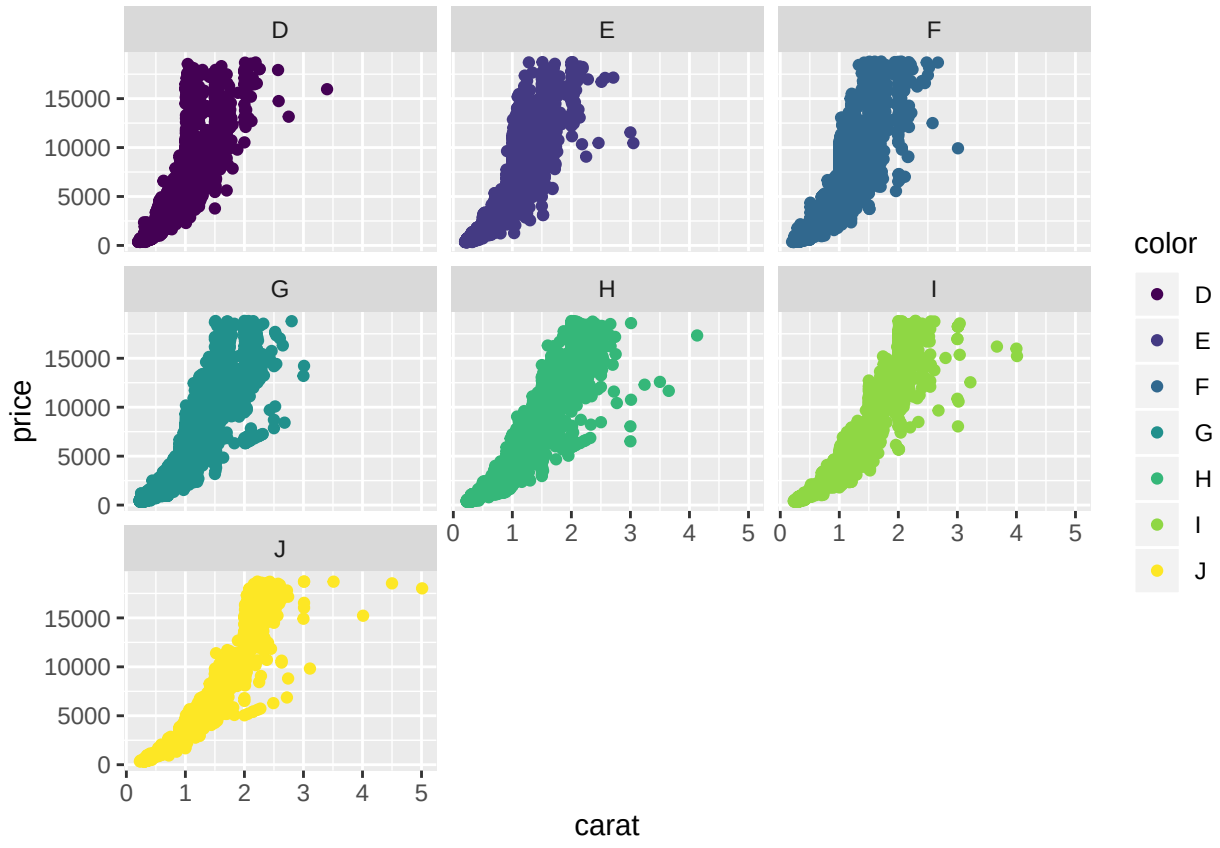

```
# density plot
ggplot(data = diamonds)+geom_density(aes(x=carat), fill="grey50")
```



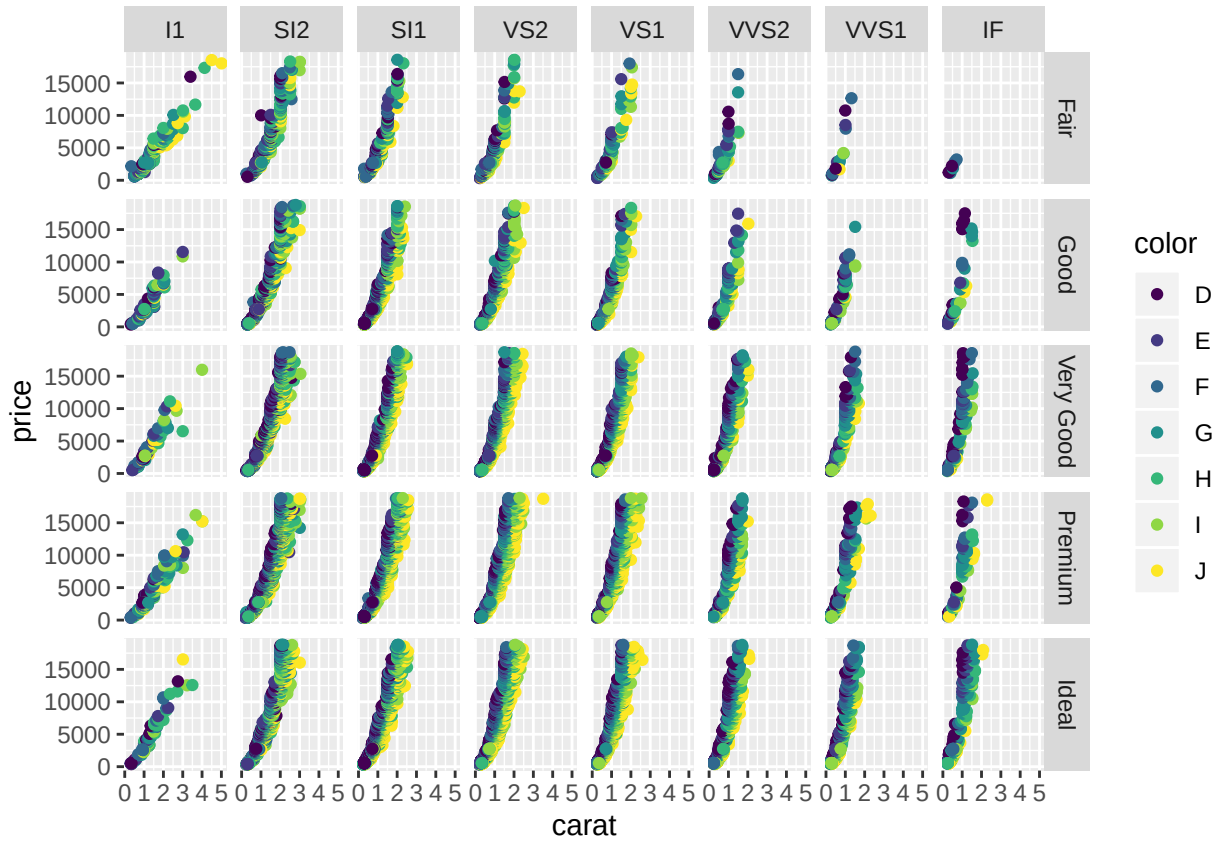

```
# more related with density : https://www.data-to-viz.com/graph/density.html

ggplot(diamonds, aes(x = carat, y = price)) + geom_point(aes(color = color))
```



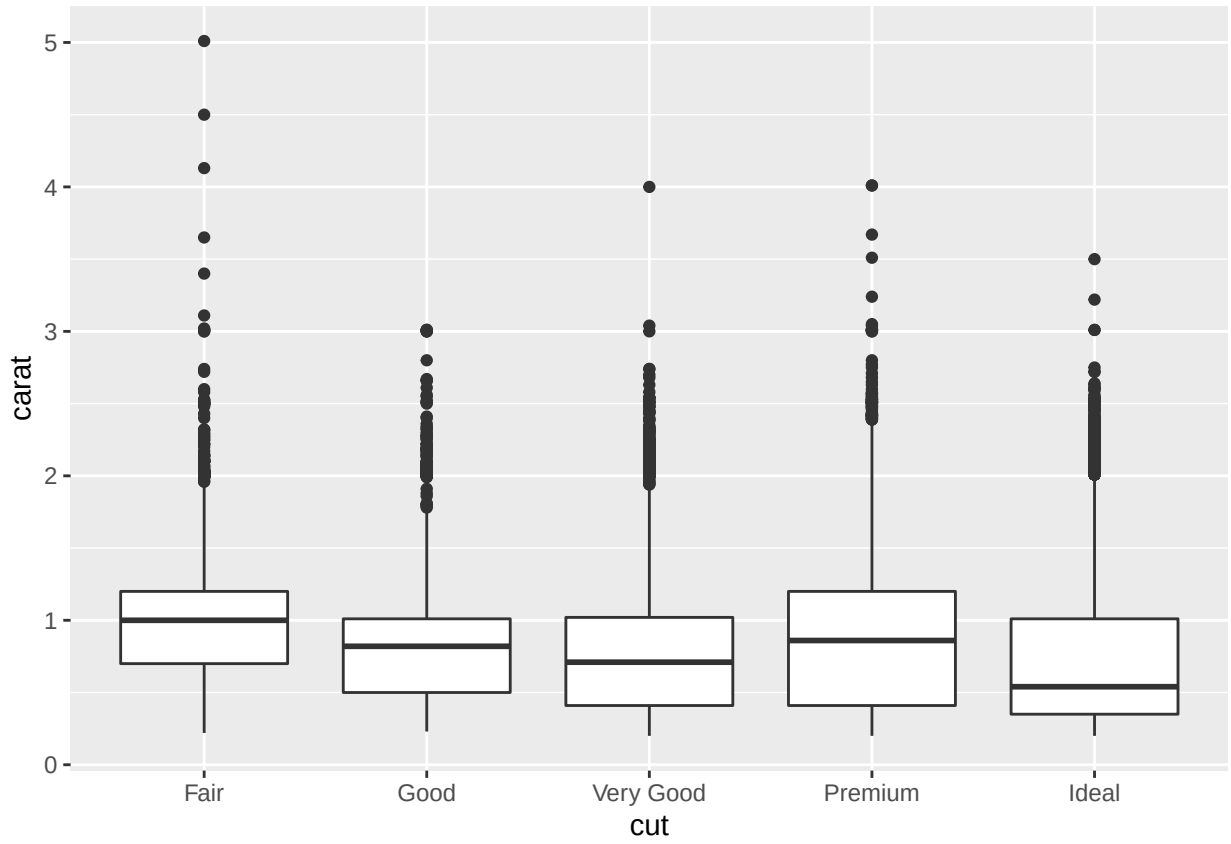

```
# facet: small part or surface of one object

ggplot(diamonds, aes(x = carat, y = price)) + geom_point(aes(color = color)) + facet_wrap(~color)
```



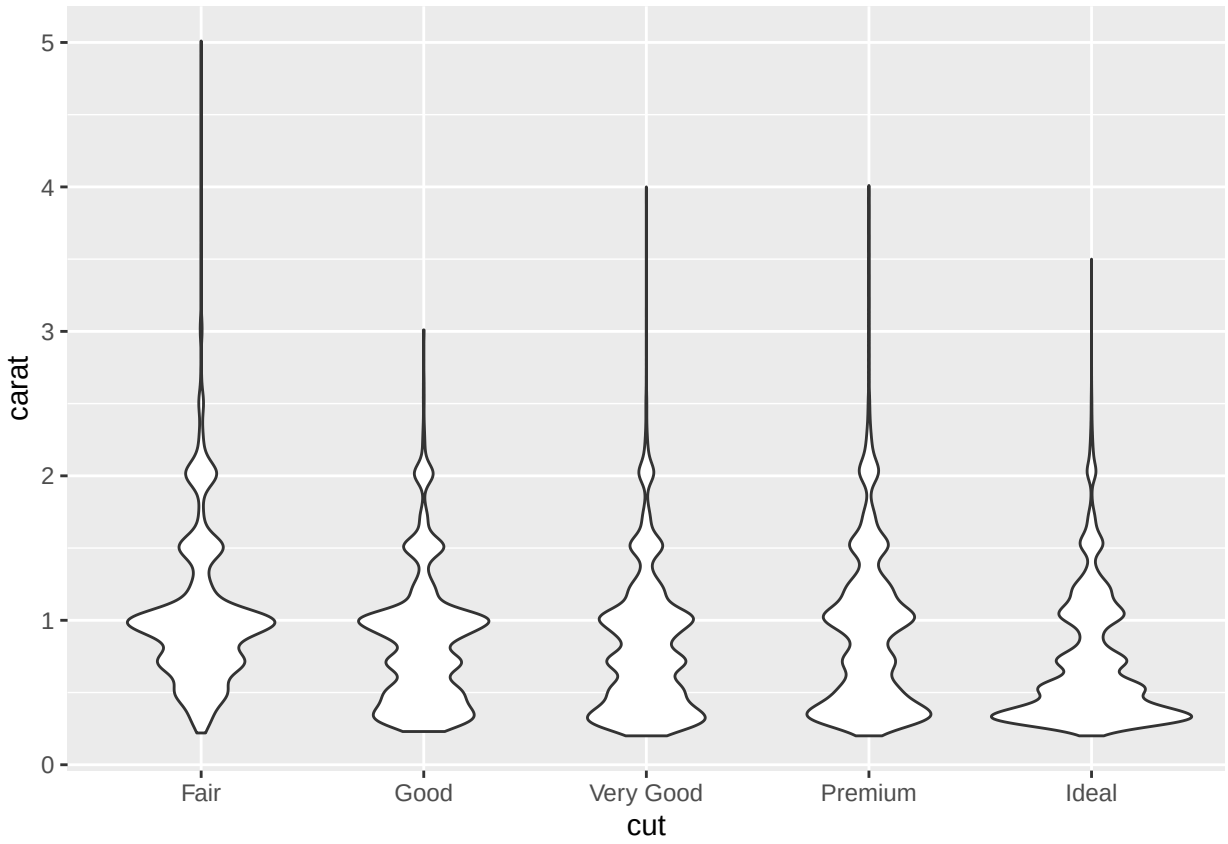

```
ggplot(diamonds, aes(x = carat, y = price)) + geom_point(aes(color = color)) + facet_grid( cut ~ clarity
```




```
ggplot(diamonds, aes(y = carat, x = cut)) +geom_boxplot()
```




```
ggplot(diamonds, aes(x = cut, y = carat)) + geom_violin()
```



Functions

```
say.hello <- function()
  {
  print("Hello, World!")
}

say.hello()
```

```
## [1] "Hello, World!"
```

```
# substituion

sprintf("Hello %s", "Pengcheng")
```

```
## [1] "Hello Pengcheng"
```

```
sprintf("Hello %s, today is %s", "Pengcheng", "Saturday")
```

```
## [1] "Hello Pengcheng, today is Saturday"
```

```
# write into one function

hello.name <- function(name)
{
  sprintf("Hello, %s", name)
}

hello.name("XiaoPangpang")
```

```
## [1] "Hello, XiaoPangpang"
```

```r
# function return values
# The value of the last line in the function
# will be automatically returned



double.num <- function(x)
{
  x*2
}
double.num(12)
```

```
## [1] 24
```

```r
# return command more explicitly addresses this issue
# and function should be exited after the return

double.sum1 <- function(x)
{
  return(2*x)

  # below will not be executed since function already exited
  # after the first return

  print("hello ")
  return(12)
}

double.sum1(11)
```

```
## [1] 22
```

```r
# do.call use:This allows us to specify the name of a function either as a character or as an object,
# and provide arguments as a list

do.call("hello.name",args = list("Pengcheng"))
```

```
## [1] "Hello, Pengcheng"
```

```r
do.call(hello.name,args = list("Niuniu"))
```

```
## [1] "Hello, Niuniu"
```

```r
run.this <- function(x, func = mean)
{
  do.call(func,args = list(x))
}

run.this(1:100)
```

```
## [1] 50.5
```

```r
run.this(1:100,sum)
```

```
## [1] 5050
```

```r
# Control statement if
```

```r
check.if1 <- function(x)
{
  if (x ==1)
  {
    print("That is correct!")
  }else
  {
    print("Try it again!")
  }
}

check.if1(1)
```

```
## [1] "That is correct!"
```

```r
check.if1("lalala")
```

```
## [1] "Try it again!"
```

```r
check.if2 <- function(x)
{
  if(x ==1)
  {
    print("yay!")
  }else if (x == 2)
  {
    print("Not bad!")
  }else
  {
    print("Come on!")
  }
}

check.if2(2)
```

```
## [1] "Not bad!"
```

```r
check.if2(3)
```

```
## [1] "Come on!"
```

```r
# Switch statement
use.switch <- function(x)
{
  switch (x,
    "a" = "action",  # first argument is the value we are testing
    "b" = "boyfriend", # on the right hand side is the corresponding results
    "c" = "charlie",
    "others"
  )
}

use.switch(1)
```

```
## [1] "action"
```

```r
use.switch(4)
```

```
## [1] "others"
```

```
use.switch(5)
is.null(switch(5))
```

```
## Warning in switch(5): 'switch' with no alternatives
```

```
## [1] TRUE
```

```
# ifelse statement , very useful

# first argument: condition to be tested
# second argument: if test is TRUE, return it
# third argument: if test is FALSE, return it

ifelse(1==1, "yes","no")
```

```
## [1] "yes"
```

```
testdata <- c(1,0,1,1,0,1,1)
ifelse(testdata == 1, "Y","N")
```

```
## [1] "Y" "N" "Y" "Y" "N" "Y" "Y"
```

```
ifelse(testdata == 1, testdata*7, testdata)
```

```
## [1] 7 0 7 7 0 7 7
```

```
# if testdata has NA elements. return result from ifelse still is NA

# MAKE SURE the first argument is == instead of =!!! Too many mistakes here!
testdata[3] <- NA
ifelse(testdata == 1, "Y","N")
```

```
## [1] "Y" "N" NA  "Y" "N" "Y" "Y"
```

```
# Compound Test
# and & &&
# or |  ||
# double form is best used in if statement
# single form is necessary in ifelse statement

a <- c(1,0,0,1)
b <- c(1,2,3,1)

# check each element from a and b
ifelse(a ==1 & b ==1, "Yay!","Oh No!")
```

```
## [1] "Yay!"   "Oh No!" "Oh No!" "Yay!"
```

```
# check only the first element in a and b
ifelse(a ==1 && b == 1, "Yay!", "Oh No!")
```

```
## [1] "Yay!"
```

```
# Single form, both sides will be checked
# double form, sometimes only left side will be checked
# 1==0 && 2==2, after checking lHS, no need to check RHS

# Loops
```

```r
for (i in 1:10)
{
  print(i)
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
## [1] 10
```

```r
family <- c("Pang", "Niu", "Xi")

familylength <- rep(NA, length(family))

names(familylength) <- family

# loop through family to assign lengths to result vector
for (a in family)
{
  familylength[a] <- nchar(a)
}

familylength
```

```
## Pang  Niu   Xi
##    4    3    2
```

```r
# While loop
x <- 2
while(x <= 7){
  print(x)
  x <- x+1
}
```

```
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
```

```r
# Control Loop, if we want to skip to next iteration
# use next or break

for ( i in 1:10)
{
  if (i == 3)
  {
    next # all number except 3
  }
```

```
  print(i)
}
```

```
## [1] 1
## [1] 2
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
## [1] 10
```

```
# stop at 8

for (i in 1:100)
{
  if (i ==8)
  {
    break
  }
  print(i)
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
```