# An R Lecture from Practice

Fangda Fan

2016.5

# Contents

# Preparation

- Bring your laptop.
- Install R **and** R-studio
- Download the dataset "Rlecture_Diamonds.csv"
- Be ready to typing in code!

# Our Goals

- Know what is R
- Learn basic operations on R vectors
- Use R to do scientific computation
- Learn different data types in R to simplify operations

# Contents

# What is R?

- A language and environment for statistical computing and graphics
- A wide variety of statistical (linear and nonlinear modelling, classical statistical tests, time-series analysis, classification, clustering, ...) and graphical techniques, and is highly extensible.
- Free Software
- Polls, surveys of data miners, ... show that R's popularity has increased substantially in recent years. (Wikipedia)

# What can R do?

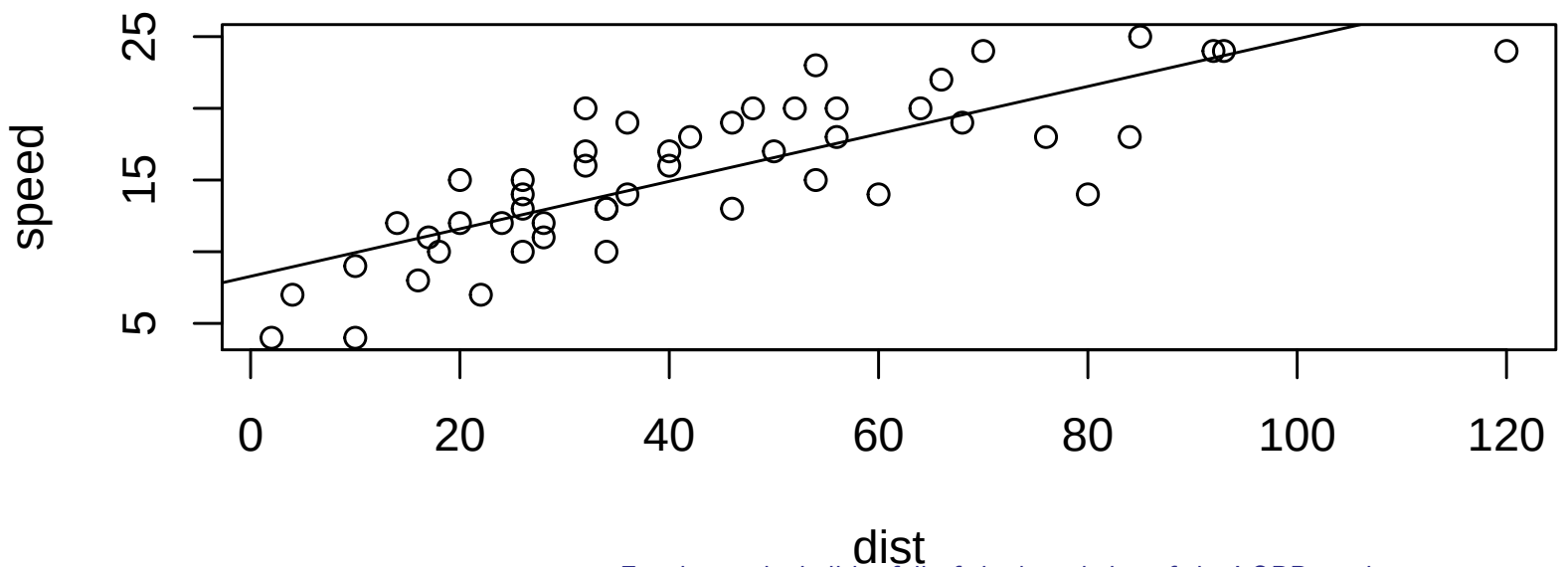
- Data: Input, output, cleaning, extracting, summarizing,...
- Statistical models: random variables and distribution, linear regression, clustering, machine learning, social analysis, ..., nearly all statistical methods that you can think out of
- Scientific computing: vector and matrix operations, lots of mathematical functions, ...
- Programming: if, for, while, user-defined functions, ...
- Presentation: various kinds of plots, LaTeX reports with "knitr"
- And more ...

# How to begin?

1. Open the RStudio
2. Create a new script from File $\rightarrow$ New File $\rightarrow$ R Scripture
3. Type the code in the upper-left area
4. Run each line with Ctrl+Enter after finishing it

# Try the power of one-line R Code

```r
## Scatter plot with an one-line code
plot(speed ~ dist, data = cars)
## Linear Regression with an one-line code!
model = lm(speed ~ dist, data = cars)
## Summery with an one-line code!
summary(model)
## Add a line with an one-line code
abline(model)
```



# Interpretation

```
summary(model)
```

```
##
## Call:
## lm(formula = speed ~ dist, data = cars)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -7.5293 -2.1550  0.3615  2.4377  6.4179
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  8.28391    0.87438   9.474 1.44e-12 ***
## dist         0.16557    0.01749   9.464 1.49e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.156 on 48 degrees of freedom
## Multiple R-squared:  0.6511,^^IAdjusted R-squared:  0.6438
## F-statistic: 89.57 on 1 and 48 DF,  p-value: 1.49e-12
```

# Distribution and Random Variables

- We can easily get critical values of given distribution from R

```
pnorm(2, mean = 0, sd = 1) # Distribution function (normal)

## [1] 0.9772499

dnorm(2, mean = 0, sd = 1) # Density function (normal)

## [1] 0.05399097

qnorm(0.975, mean = 0, sd = 1) # Quantile function (normal)

## [1] 1.959964
```

- Generate random sample from a given distribution

```
set.seed(1234) # Set a random seed to repeat a random experiment
rnorm(5, mean = 0, sd = 1) # Random sample (normal)

## [1] -1.2070657  0.2774292  1.0844412 -2.3456977  0.4291247

runif(5, min = 0, max = 1) # Random sample (uniform)

## [1] 0.6935913 0.5449748 0.2827336 0.9234335 0.2923158

rbinom(20, size = 5, prob = 0.3) # Random sample (binomial)

##  [1] 3 1 1 1 1 1 1 0 0 1 2 1 3 2 0 1 1 1 1 1
```

# Get help

- From R: ? + function
  - ?rnorm
  - ?runif
  - ?lm
  - ?plot
- From Books:
  - The Art of R Programming, Norman Matloff
  - An Introduction to Statistical Learning with Applications in R, Gareth James
- From Google: Your Question + R

# Contents

# Vectors

- Generate 5 random variables of uniform distribution between 0 and 1

```
set.seed(12345)
x = runif(5, 0, 1)
x

## [1] 0.7209039 0.8757732 0.7609823 0.8861246 0.4564810

x + 1

## [1] 1.720904 1.875773 1.760982 1.886125 1.456481
```

- Generate continuous integers from 1 to 5

```
y = 1:5
y

## [1] 1 2 3 4 5

y * 2

## [1]  2  4  6  8 10
```

# Mathematical Operations (1)

- Four arithmetic operations

```
x + y

## [1] 1.720904 2.875773 3.760982 4.886125 5.456481

x - y

## [1] -0.2790961 -1.1242268 -2.2390177 -3.1138754 -4.5435190

x * y

## [1] 0.7209039 1.7515464 2.2829470 3.5444983 2.2824048

x / y

## [1] 0.72090390 0.43788660 0.25366078 0.22153114 0.09129619
```

- Power and square root

```
x ** 2

## [1] 0.5197024 0.7669787 0.5790941 0.7852167 0.2083749

sqrt(x)

## [1] 0.8490606 0.9358275 0.8723430 0.9413419 0.6756337
```

# Mathematical Operations (2)

- Exponential and logarithm

```
exp(x)

## [1] 2.056291 2.400731 2.140378 2.425711 1.578509

log(x)

## [1] -0.3272494 -0.1326481 -0.2731451 -0.1208977 -0.7842083

log10(x)

## [1] -0.14212263 -0.05760835 -0.11862543 -0.05250522 -0.34057733
```

- Statistics

```
sum(x)

## [1] 3.700265

c(sum(x), mean(x), var(x), sd(x)) # Combine values into a vector

## [1] 3.70026494 0.74005299 0.03024368 0.17390709

c(median = median(x), minimum = min(x), maximum = max(x)) # named vector

##    median   minimum   maximum
## 0.7609823 0.4564810 0.8861246
```

# Mathematical Operations (3)

- Quantiles and ranks

```
quantile(x)

##        0%       25%       50%       75%      100%
## 0.4564810 0.7209039 0.7609823 0.8757732 0.8861246

rank(x)

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

- Round

```
round(x, 3)

## [1] 0.721 0.876 0.761 0.886 0.456

floor(x)

## [1] 0 0 0 0 0

ceiling(x)

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

- Sort

```
sort(x)

## [1] 0.4564810 0.7209039 0.7609823 0.8757732 0.8861246

x

## [1] 0.7209039 0.8757732 0.7609823 0.8861246 0.4564810
```

# Take Values by Position Index

- Assign the sorted value of x to z

```
z = sort(x)
z

## [1] 0.4564810 0.7209039 0.7609823 0.8757732 0.8861246
```

- Get the values of assigned positions from a vector

```
z[1] # the first value

## [1] 0.456481

z[2:4] # the values of positions from 2 to 4

## [1] 0.7209039 0.7609823 0.8757732

z[c(1,4,5)] # the values of positions in 1, 4 and 5

## [1] 0.4564810 0.8757732 0.8861246

z[-c(2,3)] # The other values except of the positions in 2 and 3

## [1] 0.4564810 0.8757732 0.8861246
```

# Logical Operations (1)

- Inequalities

```
1 == 0 # Whether they are equal
```

```
## [1] FALSE
```

```
z > 0.5 # Comparasion for each elements of a vector
```

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

```
z == x # Pairwise comparasion between two vectors
```

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

```
z != x # Unequal
```

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

```
z > x # Greater than
```

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

```
z <= x # Not greater than
```

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

# Logical Operations (2)

- And (&): true if both values are true

```
TRUE & FALSE

## [1] FALSE

(z > 0.5) & (z < 0.5)

## [1] FALSE FALSE FALSE FALSE FALSE
```

- Or (||): true if either of them is true

```
TRUE | FALSE

## [1] TRUE

(z > 0.5) | (z < 0.5)

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

- Not (!): true if the origin value is false

```
!0 # in R, TRUE/FALSE can be also represented by 1/0

## [1] TRUE

!(z < 0.5)

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

# Take Values by Logical Index

- Take the values greater than 0.5

```
z[z > 0.5]

## [1] 0.7209039 0.7609823 0.8757732 0.8861246
```

- Find people with specific pets

```
people = c("Annas", "Bob", "Charles", "Darrel", "Emma")
animals = c("cat", "dog", "fish")
set.seed(123)
pets = sample(animals, 5, replace = TRUE)
pets

## [1] "cat"  "fish" "dog"  "fish" "fish"

people[pets != "fish"] # Who have a pet on the land?

## [1] "Annas"   "Charles"
```

# Contents

# Matrix (1)

- Construct a matrix

```
m = matrix(c(x, y), nrow = 5) # Construct a matrix from vector x and y
m
```

```
##           [,1] [,2]
## [1,] 0.7209039    1
## [2,] 0.8757732    2
## [3,] 0.7609823    3
## [4,] 0.8861246    4
## [5,] 0.4564810    5
```

```
dim(m) # Dimensions of a matrix
```

```
## [1] 5 2
```

```
t(m) # Transpose of a matrix
```

```
##           [,1]      [,2]      [,3]      [,4]     [,5]
## [1,] 0.7209039 0.8757732 0.7609823 0.8861246 0.456481
## [2,] 1.0000000 2.0000000 3.0000000 4.0000000 5.000000
```

# Matrix (2)

- Matrix algebra

```
m2 = t(m) %*% m # Matrix multiplication
m2

##           [,1]    [,2]
## [1,]  2.859367 10.5823
## [2,] 10.582300 55.0000

det(m2) # Determinant

## [1] 45.2801

solve(m2) # Inverse of a matrix

##             [,1]        [,2]
## [1,]  1.2146618 -0.23370755
## [2,] -0.2337076  0.06314843

solve(m2, c(12, 16)) # Solve x from Ax = b

## [1] 10.836620 -1.794116
```

# Matrix (3)

- Matrix selection

```
m[3,] # Take the 3rd row

## [1] 0.7609823 3.0000000

m[,2] # Take the 2nd column

## [1] 1 2 3 4 5

m[3,2] # Take the element at row = 3 and column = 2

## [1] 3

m[2:4,c(1,2)] # Take the submatrix from row 2 to 4 and column 1, 2

##           [,1] [,2]
## [1,] 0.8757732    2
## [2,] 0.7609823    3
## [3,] 0.8861246    4
```

# List

- A list: a set of pairs of key-value

```
a = list(index = 1:5, name = people)
a

## $index
## [1] 1 2 3 4 5
##
## $name
## [1] "Annas"   "Bob"     "Charles" "Darrel"  "Emma"

a["name"] # A sub-list, not a vector!

## $name
## [1] "Annas"   "Bob"     "Charles" "Darrel"  "Emma"

a[["name"]] # This is a right way to take a value.

## [1] "Annas"   "Bob"     "Charles" "Darrel"  "Emma"

a$name # This is another right way to take a value.

## [1] "Annas"   "Bob"     "Charles" "Darrel"  "Emma"
```

# Data Frame (1)

- A Data Frame: A special list in a matrix form

```
a$animals = pets # Add a new element to a list
a

## $index
## [1] 1 2 3 4 5
##
## $name
## [1] "Annas"   "Bob"     "Charles" "Darrel"  "Emma"
##
## $animals
## [1] "cat"  "fish" "dog"  "fish" "fish"

data = data.frame(a)
data

##   index    name animals
## 1     1   Annas     cat
## 2     2     Bob    fish
## 3     3 Charles     dog
## 4     4  Darrel    fish
## 5     5    Emma    fish
```

# Data Frame (2)

- You can use both matrix and list operations on a data frame, and each column is a list element

```
data[c("index", "animals")] # Subset columns by name

##   index animals
## 1     1     cat
## 2     2    fish
## 3     3     dog
## 4     4    fish
## 5     5    fish

data$index # Select a column as a list

## [1] 1 2 3 4 5

data[2:4, 1:2] # Select rows and columns as a matrix

##   index    name
## 2     2     Bob
## 3     3 Charles
## 4     4  Darrel
```

# Marginal Apply

- apply: apply a function to a matrix by row/column

```
sum(m)

## [1] 18.70026

apply(m, 1, sum) # Apply a function by row

## [1] 1.720904 2.875773 3.760982 4.886125 5.456481

apply(m, 2, sum) # Apply a function by column

## [1]  3.700265 15.000000
```

- sapply: apply a function to a vector/list by element

```
sapply(a, length) # Apply a function by list element

##   index    name animals
##       5       5       5

sapply(data, length) # Apply a function by column of data frame

##   index    name animals
##       5       5       5
```

# Contents

# Preparation

- Download the dataset "RIecture_Diamonds.csv"

# Our Goals

- Read real data from files
- Analyze data with R Data Frame
- Learn linear regression in R

# Contents

# Read Data

Before reading data, we must set the working directory to find the data file:

1. open the "files" tag in the bottom-right area
2. Find the button "···" on the right
3. Select the folder where our data files are
4. Find the "More" button and choose "Set as working directory"
5. Then read data by read.csv()

```
data = read.csv("Rlecture_Diamonds.csv")
dim(data) # check the dimension of data

## [1] 53940    11
```

# Summarize Data

- We can summarize the data with str() and summary()

```
str(data) # Structure of data

## 'data.frame':^^I53940 obs. of  11 variables:
##  $ n      : 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    : Factor w/ 5 levels "Fair","Good",..: 3 4 2 4 2 5 5 5 1 5 ...
##  $ color  : Factor w/ 7 levels "D","E","F","G",..: 2 2 2 6 7 7 6 5 2 5 ...
##  $ clarity: Factor w/ 8 levels "I1","IF","SI1",..: 4 3 5 6 4 8 7 3 6 5 ...
##  $ 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  NA 326 NA 334 335 NA NA 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 ...

summary(data) # Summary of data

##        n             carat               cut        color
##  Min.   :    1   Min.   :0.2000   Fair     : 1610   D: 6775
##  1st Qu.:13486   1st Qu.:0.4000   Good     : 4906   E: 9797
##  Median :26971   Median :0.7000   Ideal    :21551   F: 9542
##  Mean   :26971   Mean   :0.7979   Premium  :13791   G:11292
##  3rd Qu.:40455   3rd Qu.:1.0400   Very Good:12082   H: 8304
##  Max.   :53940   Max.   :5.0100                     I: 5422
##                                                     J: 2808
##     clarity          depth           table           price
##  SI1    :13065   Min.   :43.00   Min.   :43.00   Min.   :  326
##  VS2    :12258   1st Qu.:61.00   1st Qu.:56.00   1st Qu.:  949
##  SI2    : 9194   Median :61.80   Median :57.00   Median : 2397
##  VS1    : 8171   Mean   :61.75   Mean   :57.46   Mean   : 3928
```

# Summarize Discrete Variables

- We can summarize discrete variables by frequency with table()

```
table(data$color) # Univariate frequency table

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

table(data$color, data$cut) # Cross frequency table

##
##     Fair Good Ideal Premium Very Good
##   D  163  662  2834    1603      1513
##   E  224  933  3903    2337      2400
##   F  312  909  3826    2331      2164
##   G  314  871  4884    2924      2299
##   H  303  702  3115    2360      1824
##   I  175  522  2093    1428      1204
##   J  119  307   896     808       678
```

# Summarize Continuous Variables

- We can summarize continuous variables by statistics and cut()

```
mean(data$carat) # Mean value

## [1] 0.7979397

sd(data$carat) # Standard error

## [1] 0.4740112

quantile(data$carat) # Quantiles

##   0%  25%  50%  75% 100%
## 0.20 0.40 0.70 1.04 5.01

table(cut(data$carat, breaks = 10)) # Frequency counts of equal-length cut

##
## (0.195,0.681]  (0.681,1.16]   (1.16,1.64]   (1.64,2.12]    (2.12,2.6]
##         25155         18626          7129          2349           614
##    (2.6,3.09]   (3.09,3.57]   (3.57,4.05]   (4.05,4.53]   (4.53,5.01]
##            53             6             5             2             1
```

# Summarize by Group

- We can summarize data by groups with aggregate() and xtabs()

```
aggregate(carat ~ cut, data, mean) # summarize by groups

##         cut     carat
## 1      Fair 1.0461366
## 2      Good 0.8491847
## 3     Ideal 0.7028370
## 4   Premium 0.8919549
## 5 Very Good 0.8063814

group = aggregate(carat ~ color + cut, data, mean) # group by multi-index with "+" in formula
xtabs(carat ~ color + cut, data = group) # expand a 2-D multi-index to a matrix

##       cut
## color      Fair      Good     Ideal   Premium Very Good
##     D 0.9201227 0.7445166 0.5657657 0.7215471 0.6964243
##     E 0.8566071 0.7451340 0.5784012 0.7177450 0.6763167
##     F 0.9047115 0.7759296 0.6558285 0.8270356 0.7409612
##     G 1.0238217 0.8508955 0.7007146 0.8414877 0.7667986
##     H 1.2191749 0.9147293 0.7995249 1.0164492 0.9159485
##     I 1.1980571 1.0572222 0.9130291 1.1449370 1.0469518
##     J 1.3411765 1.0995440 1.0635937 1.2930941 1.1332153
```

# Define a function: find NA values

- We learn to define a function to find NA values in variables

```
is.na(c(NA, 0, "a", "", NA))

## [1]  TRUE FALSE FALSE FALSE  TRUE

countNA = function(x){
  return(sum(is.na(x)))
  }
countNA(c(NA, 0, "a", "", NA))

## [1] 2

sapply(data, countNA)

##       n   carat     cut   color clarity   depth   table   price       x
##       0       0       0       0       0       0       0   14232       0
##       y       z
##       0       0
```

# Two ways to deal with NA values

- Delete the NA rows

```
data_clean = na.omit(data)
dim(data_clean)

## [1] 39708    11

sapply(data_clean, countNA)

##       n   carat     cut   color clarity   depth   table   price       x
##       0       0       0       0       0       0       0       0       0
##       y       z
##       0       0
```

- Fill NA values with the median/mean/... of the column

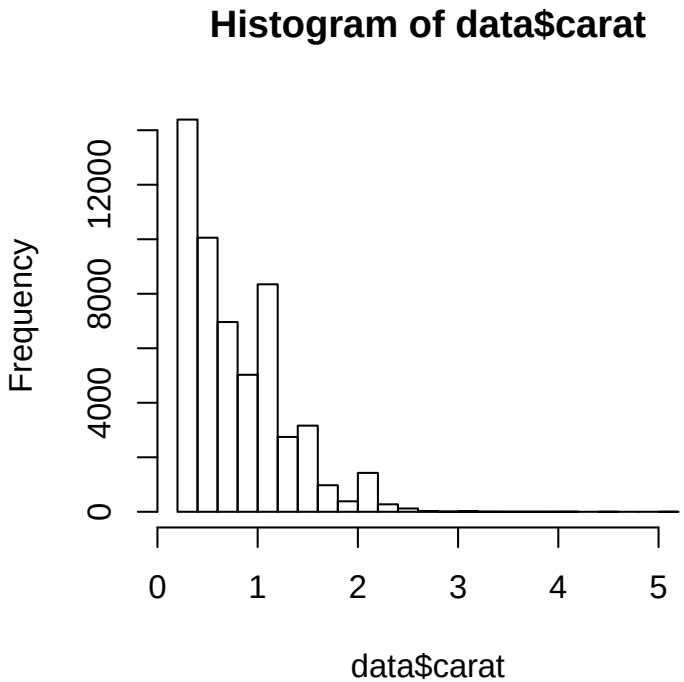
```
fillNA = function(x){
  a = median(x, na.rm = TRUE)
  x[is.na(x) == TRUE] = a
  return(x)
}
data$price = fillNA(data$price)
sapply(data, countNA)

##       n   carat     cut   color clarity   depth   table   price       x
##       0       0       0       0       0       0       0       0       0
##       y       z
##       0       0
```

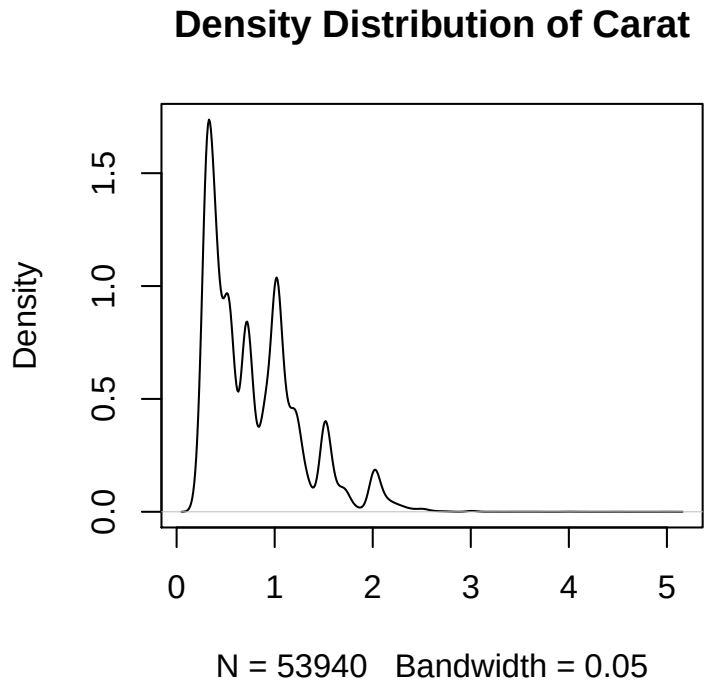
# Contents

# Univariate Plot (1)

- Visualize the distribution of numeric variables by histogram and density

```
hist(data$carat, breaks = 30)
plot(density(data$carat, width = 0.2), main = "Density Distribution of Carat")
```

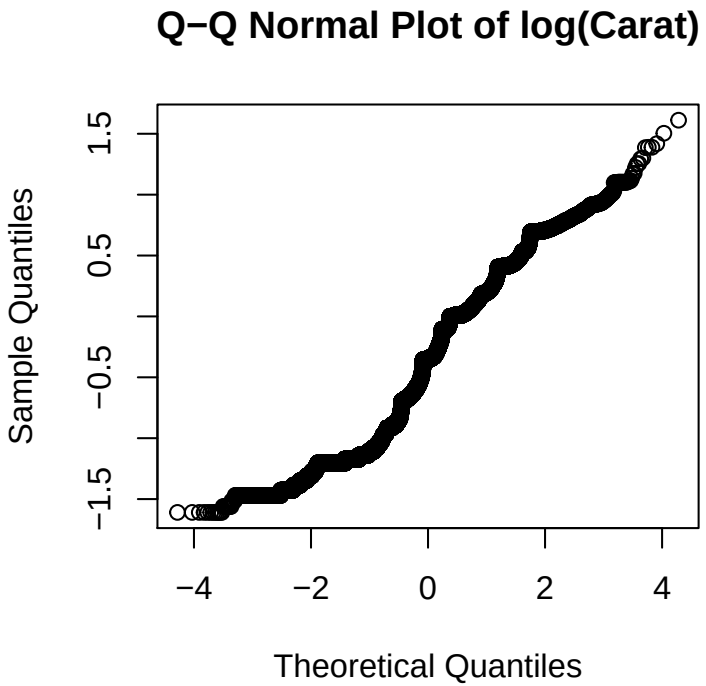


# Univariate Plot (2)

- Visualize the distribution of categorical variables by bar-plot

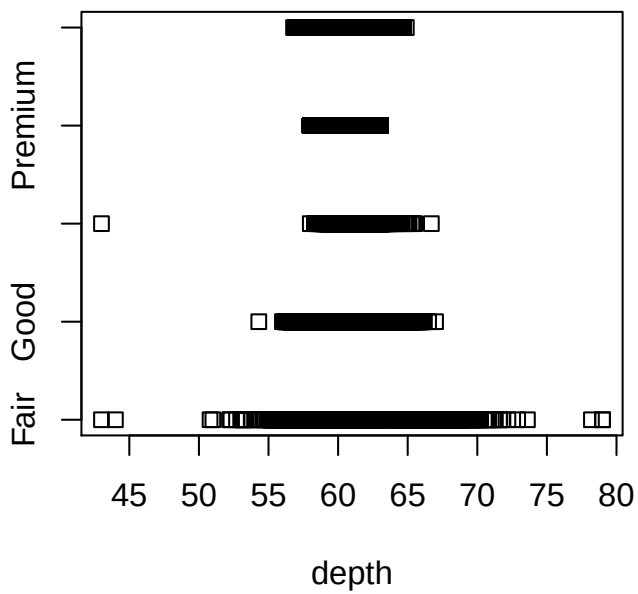
```
qqnorm(log(data$carat), main = "Q-Q Normal Plot of log(Carat)")
plot(data$color, xlab = "Color", ylab = "N", main = "Distribution of Colors")
```

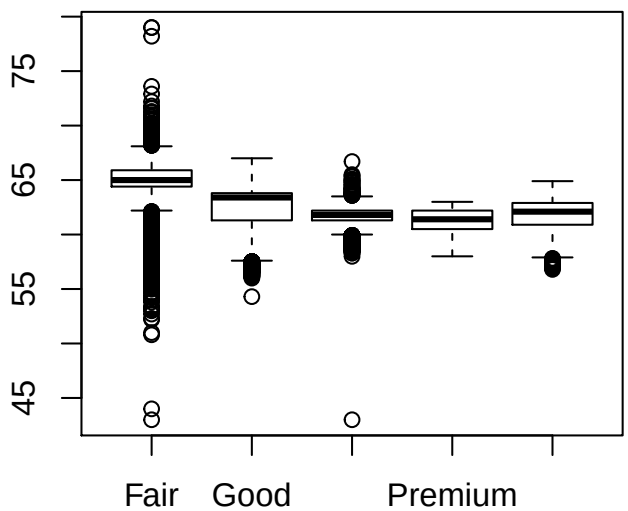



# Bivariate Plot (1)

- Plot when a numerical variable is grouped by a categorical variable

```
stripchart(depth ~ cut, data = data) # stripchart for small sample size
boxplot(depth ~ cut, data = data) # boxplot
```

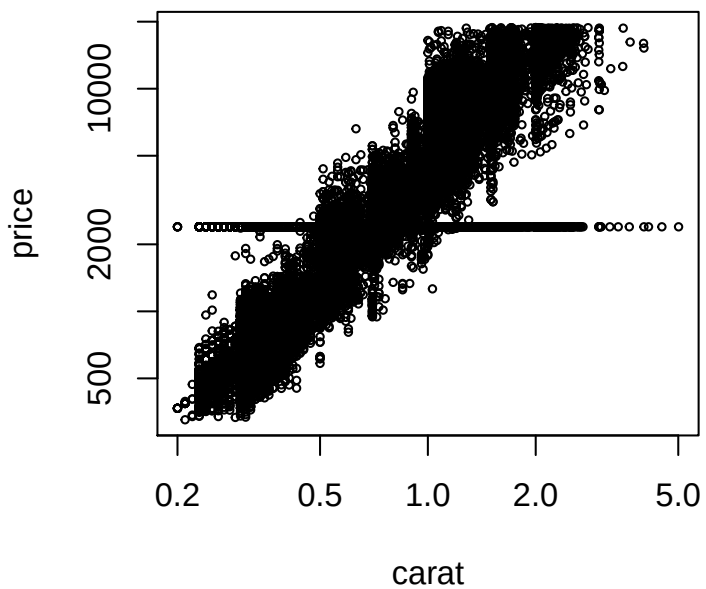


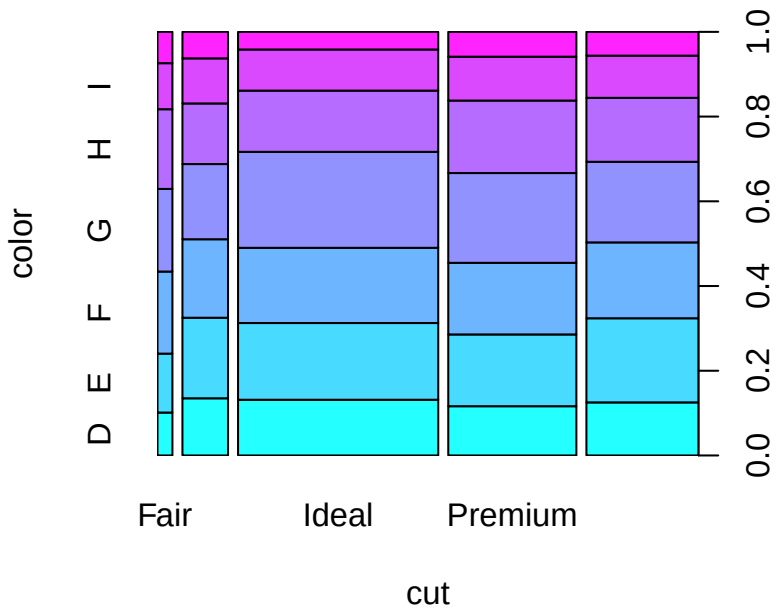



# Bivariate Plot (2)

- Function plot() can treat different types of variables automatically

```
plot(price ~ carat, data = data, cex = 0.5, log = "xy") # scatter plot for both numerical
plot(color ~ cut, data = data, col = rgb((1:7)/7, (7:1)/7, 1)) # area plot for both categorical
```



# Contents

# Theory: Linear Regression

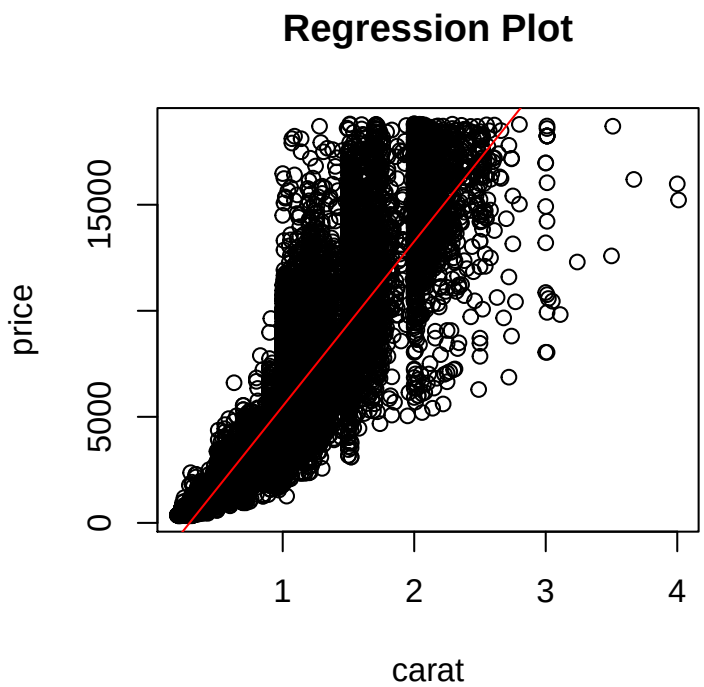
Model: $Y_{n\times 1} = \beta_0 + X_{n\times p}\beta_{p\times 1} + \varepsilon_{n\times 1}$

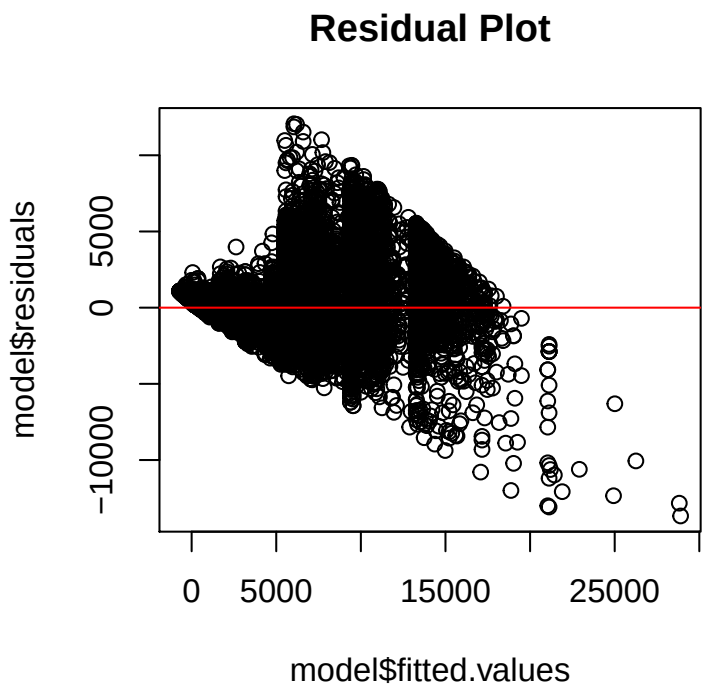
- $\varepsilon_{n\times 1} = (\varepsilon_1, \varepsilon_2, \cdots, \varepsilon_n)^T \sim^{iid} N(0, \sigma^2), \quad \sigma^2 > 0$
- Given $Y_{n\times 1} = (y_1, y_2, \cdots, y_n)^T$ and $X_{n\times p} = (X_1, X_2, \cdots, X_p)$
- Use least squares method to solve $\beta_0$ and $\beta_{p\times 1} = (\beta_1, \beta_2, \cdots, \beta_p)^T$

# Single Variable Regression

- Let Y: price, X: carat

```
model = lm(price ~ carat, data = data_clean)
plot(price ~ carat, data = data_clean, main = "Regression Plot")
abline(model, col = "red")
plot(model$residuals ~ model$fitted.values, main = "Residual Plot")
abline(0,0, col = "red")
```



# Summary of Linear Regression

- Y: price, X: carat(numeric), cut(factor) and interaction (":")

```
model = lm(price ~ carat + cut + carat:cut, data = data_clean)
summary(model)

##
## Call:
## lm(formula = price ~ carat + cut + carat:cut, data = data_clean)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -13802.3   -794.5    -21.7    545.7  12043.6
##
## Coefficients:
##                    Estimate Std. Error t value Pr(>|t|)
## (Intercept)        -2056.50     101.18 -20.325  < 2e-16 ***
## carat               6174.20      88.06  70.110  < 2e-16 ***
## cutGood             -387.22     114.00  -3.397 0.000683 ***
## cutIdeal            -249.57     103.65  -2.408 0.016053 *
## cutPremium          -312.43     105.39  -2.965 0.003033 **
## cutVery Good        -363.72     106.09  -3.428 0.000608 ***
## carat:cutGood       1315.59     103.53  12.708  < 2e-16 ***
## carat:cutIdeal      2024.48      92.17  21.964  < 2e-16 ***
## carat:cutPremium    1617.86      92.57  17.476  < 2e-16 ***
## carat:cutVery Good  1760.99      94.59  18.616  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1489 on 39698 degrees of freedom
## Multiple R-squared:  0.8609,^^IAdjusted R-squared:  0.8608
## F-statistic: 2.729e+04 on 9 and 39698 DF,  p-value: < 2.2e-16
```

# ANOVA Table

- ANOVA table

```
anova(model)

## Analysis of Variance Table
##
## Response: price
##              Df     Sum Sq    Mean Sq   F value    Pr(>F)
## carat         1 5.3925e+11 5.3925e+11 243069.75 < 2.2e-16 ***
## cut           4 4.3910e+09 1.0977e+09    494.82 < 2.2e-16 ***
## carat:cut     4 1.2771e+09 3.1927e+08    143.91 < 2.2e-16 ***
## Residuals 39698 8.8070e+10 2.2185e+06
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

- Add source to the model (“.” indicates all source in the data/model)

```
add1(model, ~ . + clarity, test = "F")

## Single term additions
##
## Model:
## price ~ carat + cut + carat:cut
##         Df  Sum of Sq        RSS    AIC F value    Pr(>F)
## <none>                8.8070e+10 580237
## clarity  7 2.5146e+10 6.2924e+10 566901    2266 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

# Stepwise Selection

- Stepwise selection from all variables from both forward and backward directions

```
model = lm(price ~ ., data = data_clean)
model_step = step(model, direction = "both", trace = 0)
anova(model_step)

## Analysis of Variance Table
##
## Response: price
##               Df     Sum Sq    Mean Sq    F value    Pr(>F)
## n              1 5.9563e+10 5.9563e+10 4.7941e+04 < 2.2e-16 ***
## carat          1 4.8098e+11 4.8098e+11 3.8712e+05 < 2.2e-16 ***
## cut            4 4.2296e+09 1.0574e+09 8.5106e+02 < 2.2e-16 ***
## color          6 9.3278e+09 1.5546e+09 1.2513e+03 < 2.2e-16 ***
## clarity        7 2.7198e+10 3.8855e+09 3.1273e+03 < 2.2e-16 ***
## depth          1 5.1445e+05 5.1445e+05 4.1410e-01    0.5199
## table          1 5.8029e+07 5.8029e+07 4.6706e+01 8.367e-12 ***
## x              1 2.3279e+09 2.3279e+09 1.8737e+03 < 2.2e-16 ***
## Residuals 39685 4.9306e+10 1.2424e+06
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

# More Transformations of Linear Model Variables

- Use "*" in formula for both main effect ("+") and interaction effect (":")

```
model_interact = lm(price ~ carat * cut * color, data = data_clean)
summary(model_interact)
```

- Use "-" for deleting source (1 indicates intercept)

```
model_reduced = lm(price ~ . - 1 - cut, data = data_clean)
summary(model_reduced)
```

- More transformation of numerical variables: cut, log, polynomial, ...

```
model_cut = lm(price ~ cut(carat, breaks = 5) + cut, data = data_clean)
summary(model_cut)
model_log = lm(log(price) ~ log(carat) + cut, data = data_clean)
summary(model_log)
model_poly = lm(price ~ poly(carat, 3) + cut, data = data_clean)
summary(model_poly)
model_poly_raw = lm(price ~ poly(carat, 3, raw = TRUE) + cut, data = data_clean)
summary(model_poly_raw)
```

# Contents

# Preparation

- Download the dataset "RIecture_Diamonds.csv"

# Our Goals

- Learn advanced data plot by ggplot2, with cheatsheet for further reference
- Learn how to choose various kinds of plot based on variable types
- Learn to plot with x, y and grouping variables in data.frame

# Contents

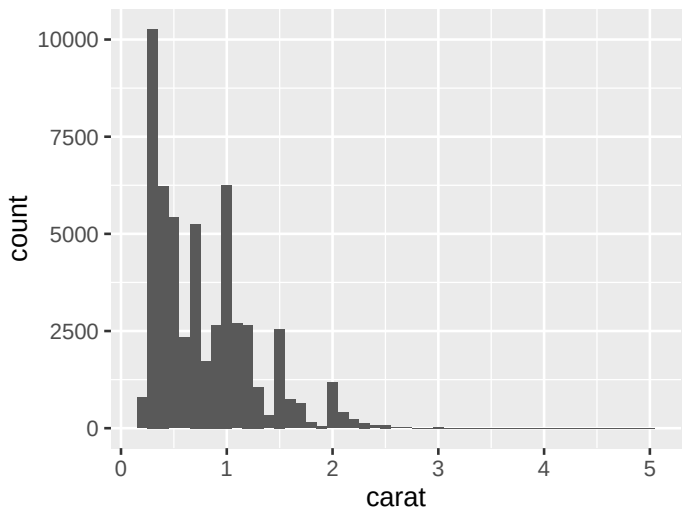
# ggplot2

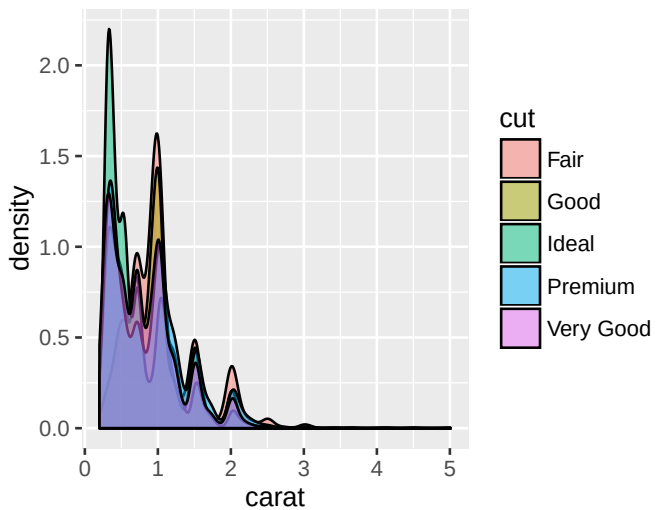
- Read data

```
data = read.csv("Rlecture_Diamonds.csv")
```

- Install by install.packages("ggplot2")

```
library("ggplot2")
ggplot(data, aes(x = carat)) + geom_histogram(binwidth = 0.1)
ggplot(data, aes(x = carat, fill = cut)) + geom_density(alpha = 0.5)
```



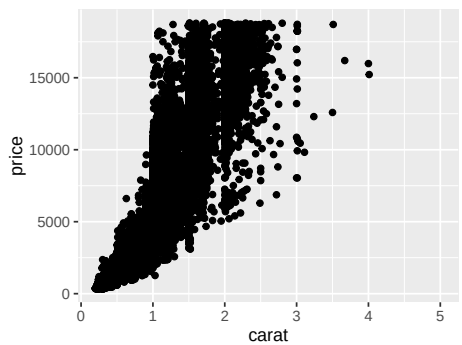



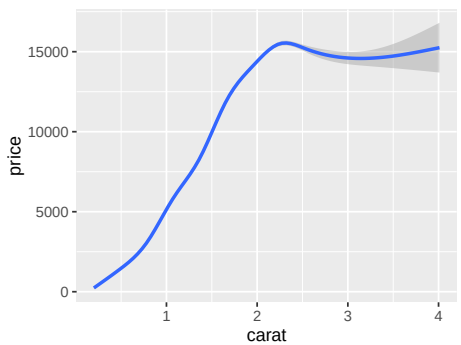
# Introduction to ggplot2

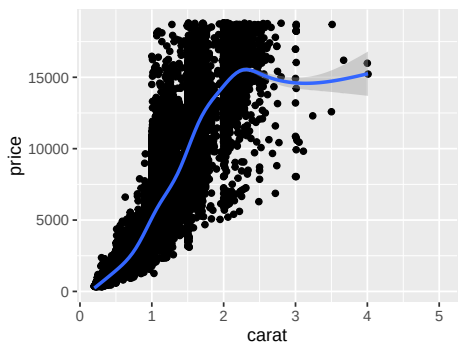
ggplot(data, mapping = aes(x = x, y = y, ...)) + geom...()

- ggplot(data, mapping): prepare data for plot
- aes(): select variables to map and group fill/color/... by variable
- geom...(): how we show the data we select in ggplot

```
ggplot(data, aes(x = carat, y = price)) + geom_point()
ggplot(data, aes(x = carat, y = price)) + geom_smooth()
ggplot(data, aes(x = carat, y = price)) + geom_point() + geom_smooth()
```



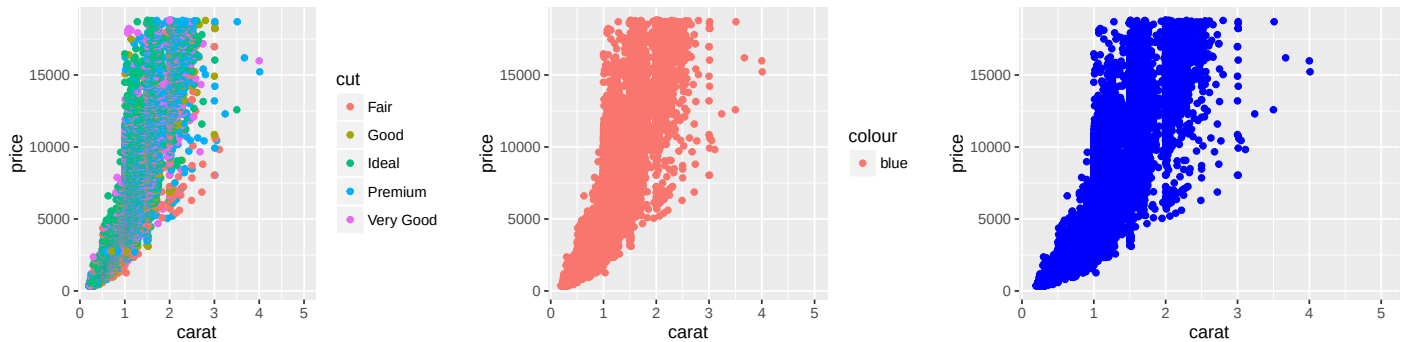



# Parameters of aes()

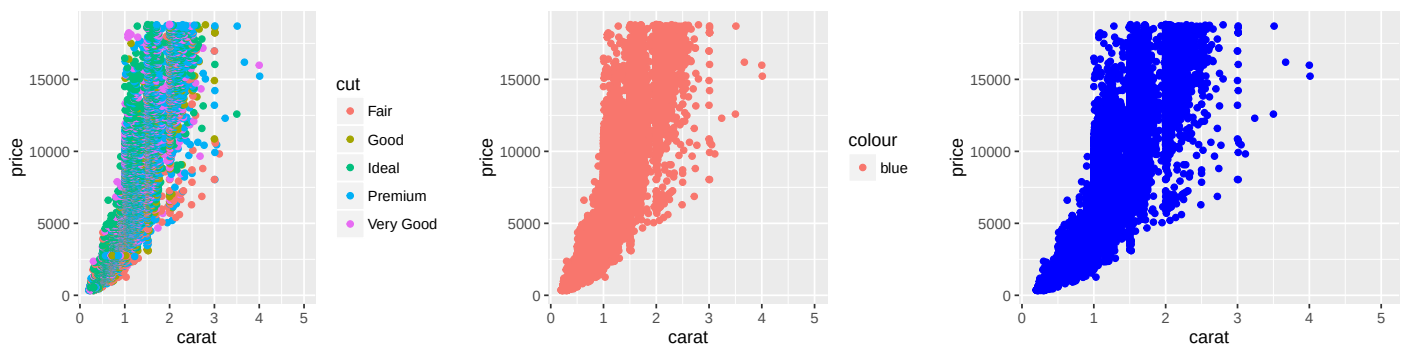
- Parameters in aes(...): use variables in data

```
ggplot(data, aes(x = carat, y = price, color = cut)) + geom_point()
ggplot(data, aes(x = carat, y = price, color = "blue")) + geom_point()
ggplot(data, aes(x = carat, y = price)) + geom_point(color = "blue")
```




- We can move aes() to anywhere

```
ggplot(data, aes(x = carat, y = price)) + geom_point(aes(color = cut))
ggplot(data, aes(x = carat)) + geom_point(aes(y = price, color = "blue"))
ggplot(data) + geom_point(aes(x = carat, y = price), color = "blue")
```

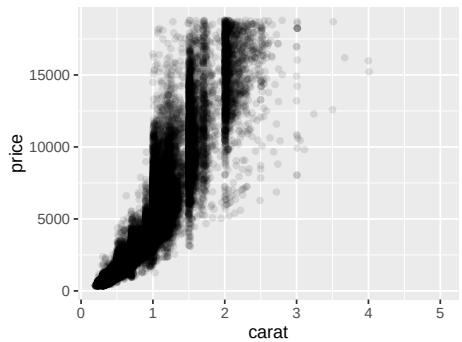


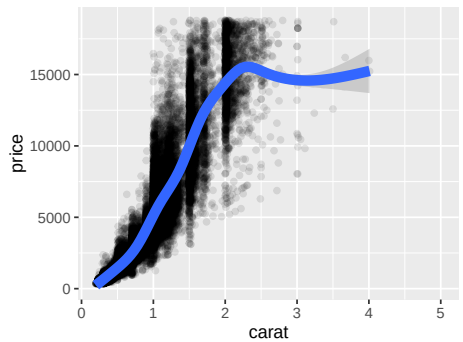

ver the sea.(Isaiah 11:9)

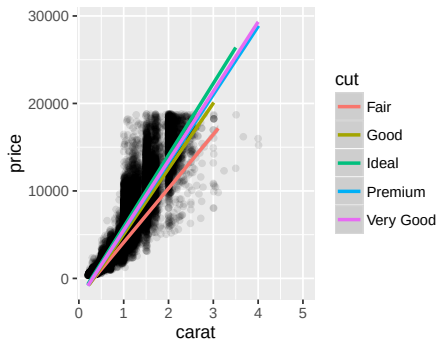
# Save Plot Elements as R Variables

- We can save ggplot variables as R lists for further use

```
p = ggplot(data, aes(x = carat, y = price))
(p = p + geom_point(alpha = 0.1))
p + geom_smooth(size = 3)
p + geom_smooth(aes(color = cut), method = "lm")
```



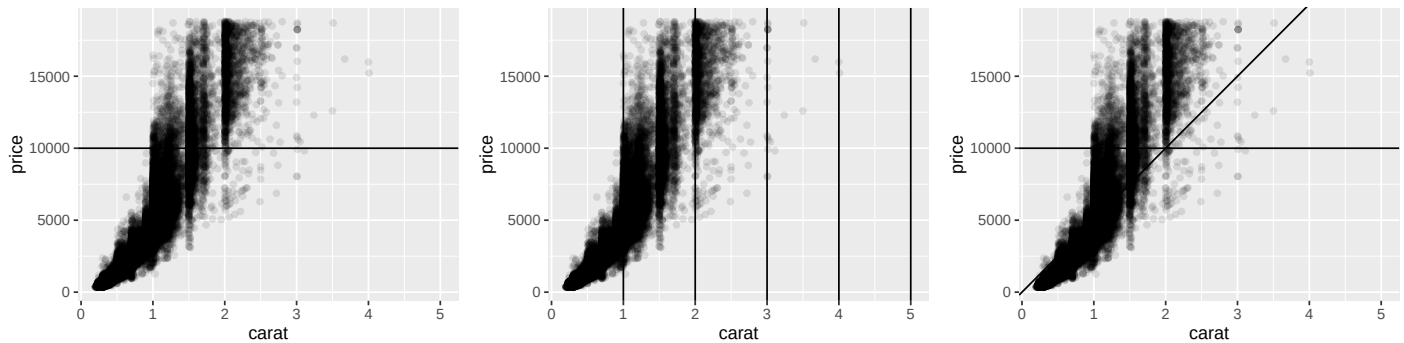



# Add Lines

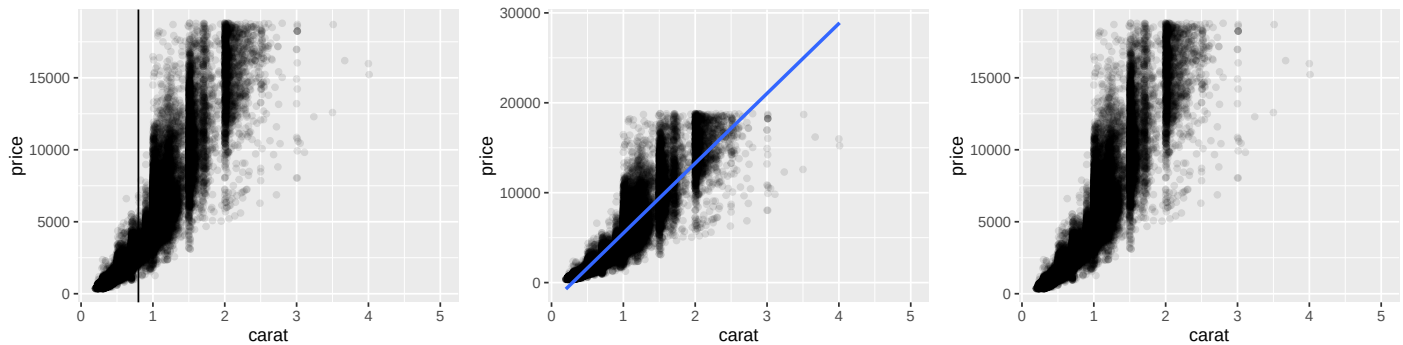
- Add straight line with geom_abline() and vertical line with geom_vline()

```
p + geom_hline(yintercept = 10000)
p + geom_vline(xintercept = 1:5)
p + geom_abline(intercept = c(0, 10000), slope = c(5000, 0))
```




- Add line with stats

```
p + geom_vline(aes(xintercept = mean(carat)))
p + geom_smooth(method = "lm", se = FALSE)
p + geom_quantile(quantiles = c(0.25, 0.5, 0.75))
```



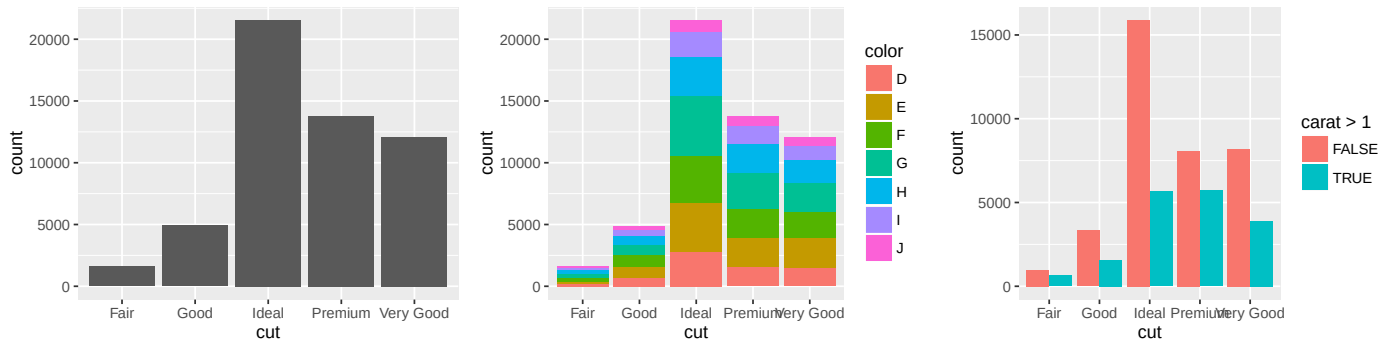

ver the sea.(Isaiah 11:9)

# Contents

# Univariate Plot (1): Discrete variable

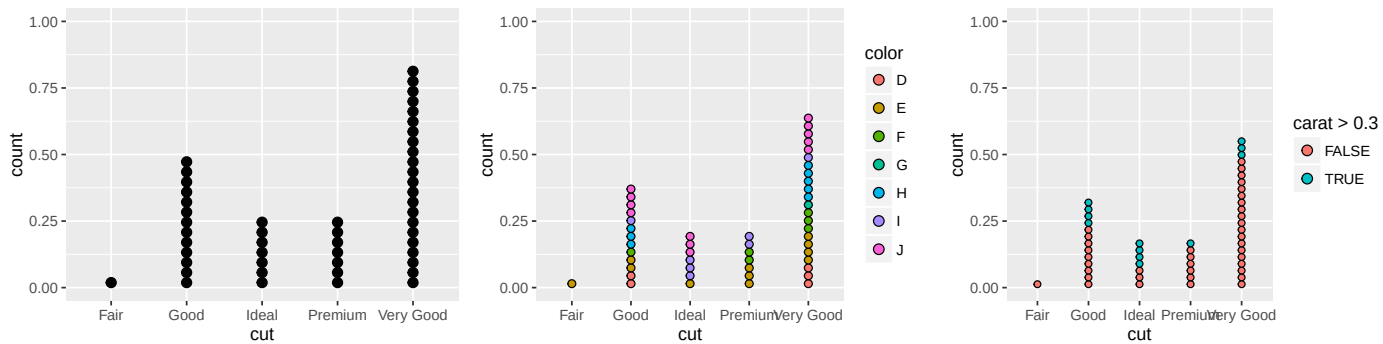
- Bar plot with geom_bar()

```
ggplot(data, aes(x = cut)) + geom_bar()
ggplot(data, aes(x = cut)) + geom_bar(aes(fill = color))
ggplot(data, aes(x = cut)) + geom_bar(aes(fill = carat > 1), position=position_dodge())
```




- Dot plot with geom_dotplot()

```
ggplot(data[1:50,], aes(x = cut)) + geom_dotplot()
ggplot(data[1:50,], aes(x = cut)) + geom_dotplot(aes(fill = color), stackgroups = TRUE)
ggplot(data[1:50,], aes(x = cut)) + geom_dotplot(aes(fill = carat > 0.3), stackgroups = TRUE)
```




ver the sea.(Isaiah 11:9)

# Univariate Plot (2): Continuous variable

- Histogram and density plot

```
ggplot(data, aes(x = depth)) + geom_histogram()
ggplot(data, aes(x = depth)) + geom_histogram(aes(fill = cut), binwidth = 1)
ggplot(data, aes(x = depth)) + geom_density()
ggplot(data, aes(x = depth)) + geom_density(aes(fill = cut), alpha = 0.5)
```


- Empirical cdf plot and quantile-quantile plot

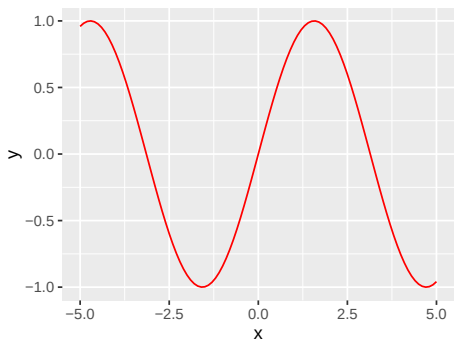
```
ggplot(data, aes(x = depth)) + stat_ecdf()
ggplot(data, aes(x = depth)) + stat_ecdf(aes(color = cut))
ggplot(data, aes(sample = depth)) + stat_qq()
ggplot(data, aes(sample = depth)) + stat_qq(aes(color = cut))
```

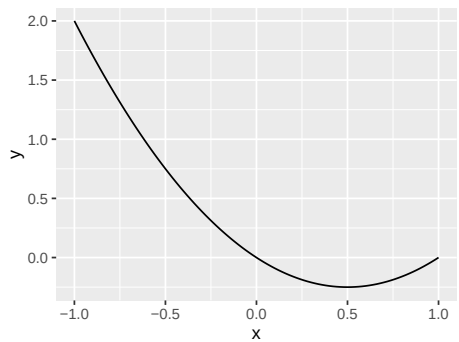

Isaiah 11:9)

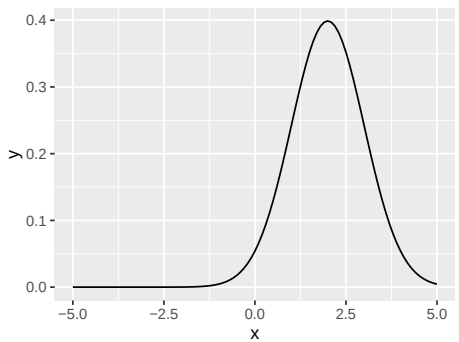
# Univariate Plot (3): Function

- We can plot univariable function using ggplot

```
ggplot(data.frame(x = c(-5, 5)), aes(x = x)) + stat_function(fun = sin, color = "red")
ggplot(data.frame(x = c(-1, 1)), aes(x = x)) + stat_function(fun = function(x){return(x^2 - x)})
ggplot(data.frame(x = c(-5, 5)), aes(x = x)) + stat_function(fun = dnorm, args = list(mean = 2))
```



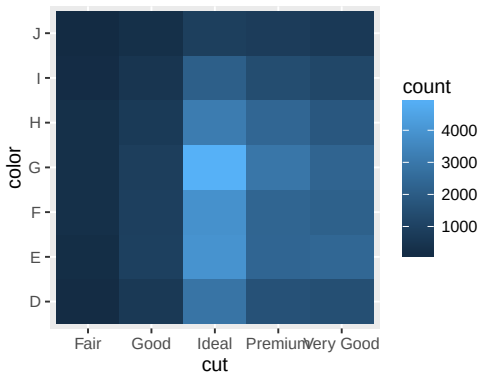



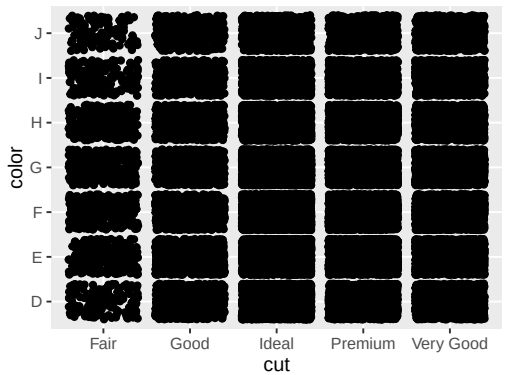



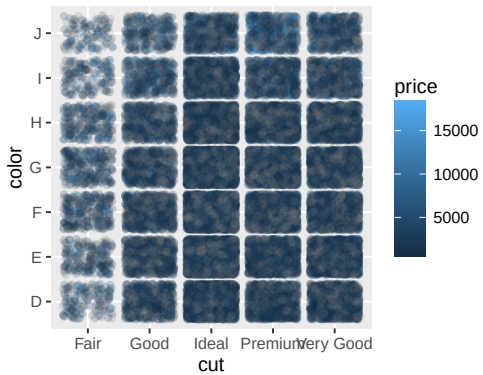
# Bivariate Plot (1): Two Discrete variables

- Take roughly count with geom_bin2d() and geom_jitter()

```
ggplot(data, aes(x = cut, y = color)) + geom_bin2d()
ggplot(data, aes(x = cut, y = color)) + geom_jitter()
ggplot(data, aes(x = cut, y = color)) + geom_jitter(aes(color = price), alpha = 0.2)
```



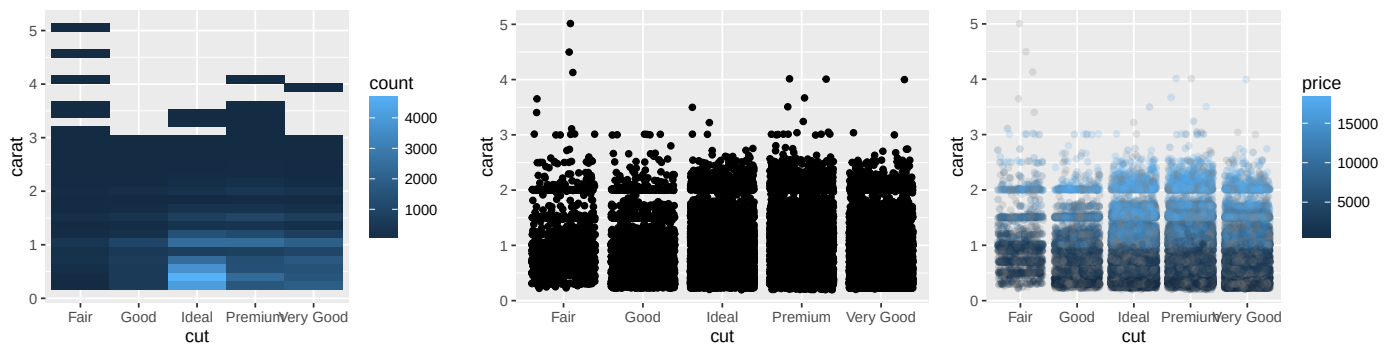
# Bivariate Plot (2): A Continuous and A Discrete

- Box plot and violin plot

```
ggplot(data, aes(x = cut, y = carat)) + geom_boxplot()
ggplot(data, aes(x = cut, y = carat)) + geom_boxplot(aes(fill = cut), varwidth = TRUE)
ggplot(data, aes(x = cut, y = carat)) + geom_violin()
ggplot(data, aes(x = cut, y = carat)) + geom_violin(aes(fill = cut), scale = "count")
```


- 2-D heatmap and jitter plot

```
ggplot(data, aes(x = cut, y = carat)) + geom_bin2d()
ggplot(data, aes(x = cut, y = carat)) + geom_jitter()
ggplot(data, aes(x = cut, y = carat)) + geom_jitter(aes(color = price), alpha = 0.2)
```

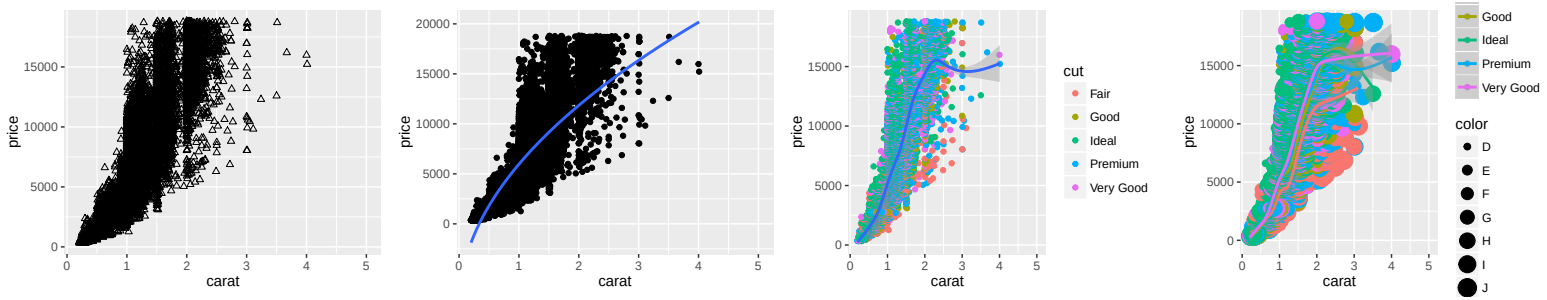



ver the sea.(Isaiah 11:9)

# Bivariate Plot (3): Two Continuous Variables

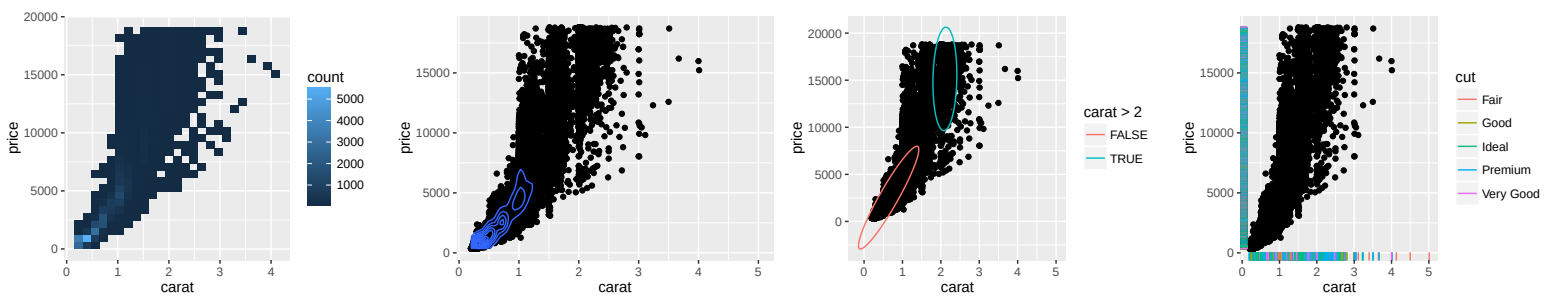
- Point plot and with smooth line

```
ggplot(data, aes(x = carat, y = price)) + geom_point(size = 1.5, shape = 2)
ggplot(data, aes(x = carat, y = price)) + geom_point()+geom_smooth(method="lm",formula=y~sqrt(x))
ggplot(data, aes(x = carat, y = price)) + geom_point(aes(color = cut)) + geom_smooth()
ggplot(data, aes(x = carat, y = price, color = cut)) + geom_point(aes(size=color))+geom_smooth()
```




- 2-D heatmap, density, ellipses and marginal rug

```
ggplot(data, aes(x = carat, y = price)) + geom_bin2d()
ggplot(data, aes(x = carat, y = price)) + geom_point() + geom_density_2d()
ggplot(data, aes(x = carat, y = price)) + geom_point() + stat_ellipse(aes(color = carat > 2))
ggplot(data, aes(x = carat, y = price)) + geom_point() + geom_rug(aes(color = cut))
```

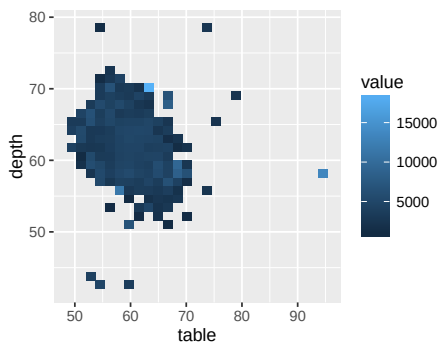


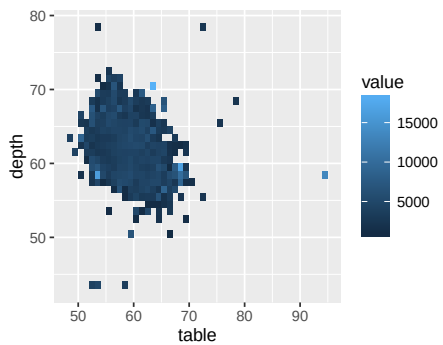

Isaiah 11:9)

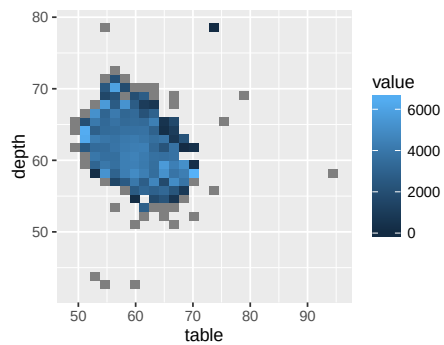
# Trivariate Plot: 2D summary

- Summary z grouped by x and y

```
ggplot(data, aes(x = table, y = depth, z = price)) + stat_summary_2d()
ggplot(data, aes(x = table, y = depth, z = price)) + stat_summary_2d(binwidth = c(1, 1))
ggplot(data, aes(x = table, y = depth, z = price)) + stat_summary_2d(fun = sd)
```



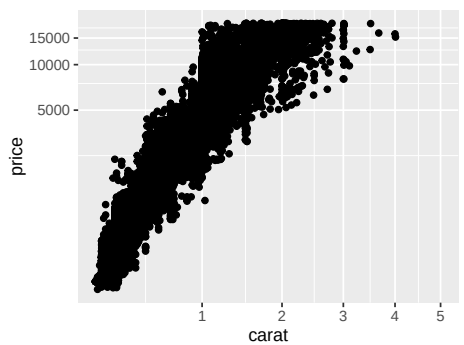



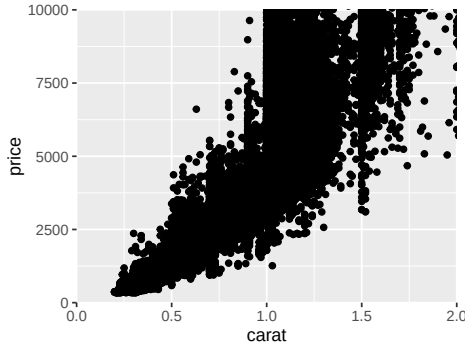
# Contents

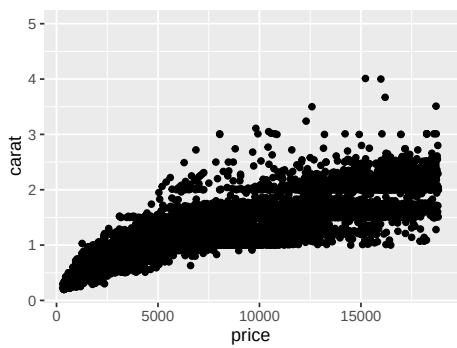
# Axes, labels and themes

- Transform and flip coordinate axes

```
p = ggplot(data, aes(x = carat, y = price)) + geom_point()
p + coord_trans(x="sqrt", y = "log10")
p + coord_trans(limx=c(0, 2), limy = c(0, 10000))
p + coord_flip()
```

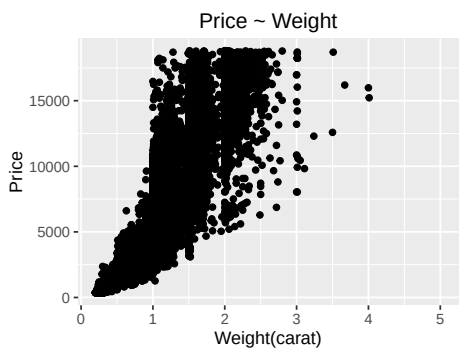


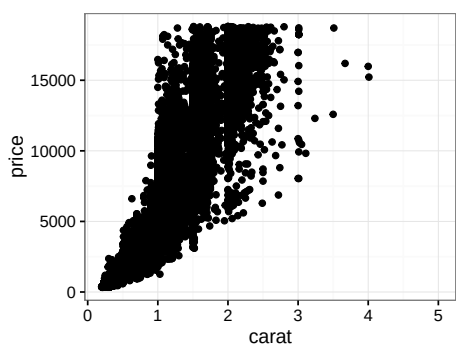



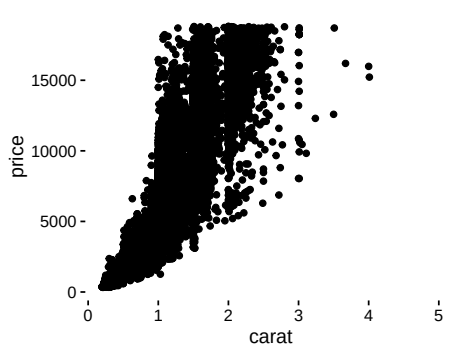




- Change title and axis labels, themes

```
p + labs(title = "Price ~ Weight", x = "Weight(carat)", y = "Price")
p + theme_bw()
p + theme_classic()
```



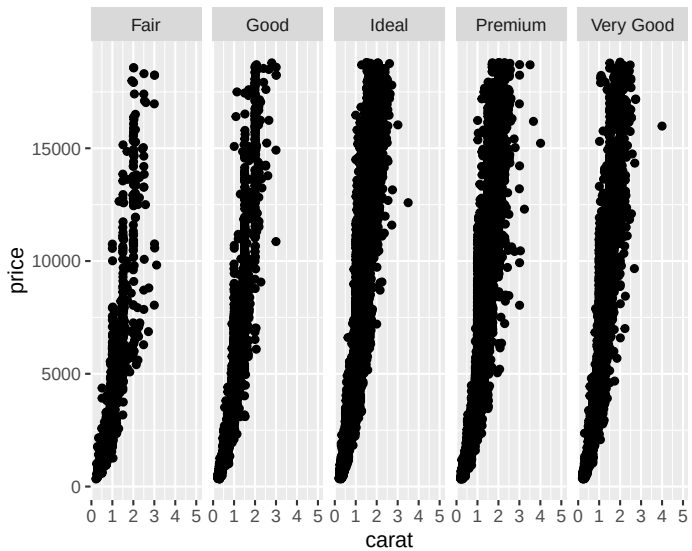



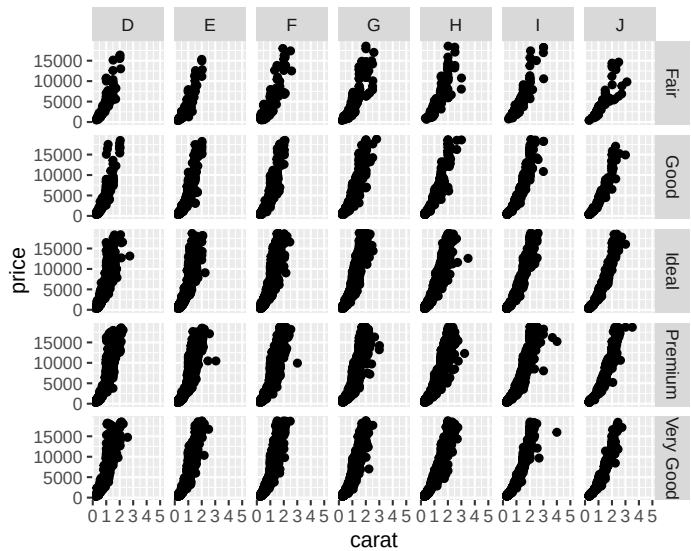

ver the sea.(Isaiah 11:9)

# Facet

- Make ggplot facet with facet_grid(facets = formula)

```
p + facet_grid(. ~ cut)
p + facet_grid(cut ~ color)
```

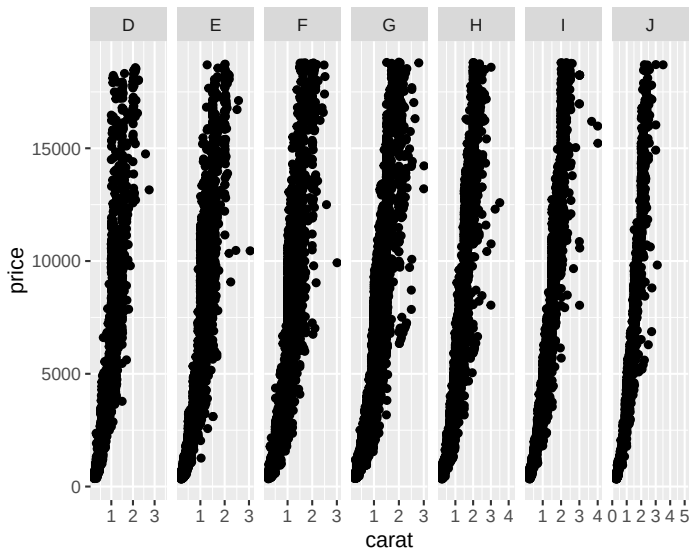


# Scales and Margins in Facet

- We can adjust scales and margins in facet_grid()

```
p + facet_grid(. ~ color, scales = "free")
p + facet_grid(cut ~ color, margins = TRUE)
```




# Contents

# Preparation

- Download the dataset "Rlecture_Diamonds.csv"

# Our Goals

- Learn data operations simplified by data.table
- Learn how to summarize information of variables for cleaning
- Learn to design steps for data cleaning in data table
- Learn how to use function/for/if and other structures in R working procedure

# Contents

# Data Table: A Poweful Extension of Data Frame

- Install by install.packages("data.table")

```
library("data.table")
data = fread("Rlecture_Diamonds.csv")
str(data)

## Classes 'data.table' and 'data.frame':^^I53940 obs. of  11 variables:
##  $ n      : 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  NA 326 NA 334 335 NA NA 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 ...
##  - attr(*, ".internal.selfref")=<externalptr>

summary(data)

##        n             carat            cut                color          
##  Min.   :    1   Min.   :0.2000   Length:53940       Length:53940       
##  1st Qu.:13486   1st Qu.:0.4000   Class :character   Class :character   
##  Median :26971   Median :0.7000   Mode  :character   Mode  :character   
##  Mean   :26971   Mean   :0.7979                                         
##  3rd Qu.:40455   3rd Qu.:1.0400                                         
##  Max.   :53940   Max.   :5.0100                                         
##                                                                         
##    clarity              depth           table           price       
##  Length:53940       Min.   :43.00   Min.   :43.00   Min.   :  326  
```

# Take Subset from Data Table

- Take subset like data.frame, but using list to contain variables

```
data[2:4, list(n, carat, price)] # select row 2 to 4 and column (n, carat, x)
```

```
##    n carat price
## 1: 2  0.21   326
## 2: 3  0.23    NA
## 3: 4  0.29   334
```

- Use .N as the length of data, and operations in columns

```
data[5:.N, list(n, unit_price = price/carat)]
```

```
##             n unit_price
##     1:      5   1080.645
##     2:      6         NA
##     3:      7         NA
##     4:      8   1296.154
##     5:      9   1531.818
##    ---
## 53932: 53936   3829.167
## 53933: 53937   3829.167
## 53934: 53938         NA
## 53935: 53939   3205.814
## 53936: 53940         NA
```

# Introduction of Data Table

- data.table has the form: data[i, j, by = ...] like SQL

```
data[i = price > 1000, j = list(count = .N, carat = mean(carat)), by = list(cut)]
```

```
##          cut count     carat
## 1:      Fair  1114 1.0813914
## 2:     Ideal 10814 0.8757287
## 3: Very Good  6507 0.9784555
## 4:      Good  2795 0.9967263
## 5:   Premium  7794 1.0605799
```

| R (data.table) | i | j | by |
|---|---|---|---|
| SQL | WHERE | SELECT | GROUP BY |

- data.table is faster than R default, especially with big data

```
system.time(read.csv("Rlecture_Diamonds.csv")) # timing (second): read csv with R default
```

```
##    user  system elapsed
##    0.25    0.00    0.25
```

```
system.time(fread("Rlecture_Diamonds.csv")) # timing (second): read csv with data.table
```

```
##    user  system elapsed
##    0.22    0.00    0.22
```

# Sort Data Table

- Sort data by setkey()

```
setkey(data, cut)
data[,list(n,cut)]
```

```
##             n       cut
##     1:      9      Fair
##     2:     92      Fair
##     3:     98      Fair
##     4:    124      Fair
##     5:    125      Fair
##    ---
## 53936: 53922 Very Good
## 53937: 53923 Very Good
## 53938: 53933 Very Good
## 53939: 53934 Very Good
## 53940: 53938 Very Good
```

- Select rows by key value directly

```
data["Good", .N]
```

```
## [1] 4906
```

```
data["Good", mult = "first"]
```

```
##    n carat  cut color clarity depth table price    x    y    z
## 1: 3  0.23 Good     E     VS1  56.9    65    NA 4.05 4.07 2.31
```

# Column Apply in Data Table

- Apply functions into each column with lapply and .SD

```
class(data[, price])

## [1] "integer"

data[,lapply(.SD, class)]

##          n   carat       cut     color   clarity   depth   table   price
## 1: integer numeric character character character numeric numeric integer
##          x       y       z
## 1: numeric numeric numeric
```

- Take a subset of columns with .SDcols

```
data[,lapply(.SD, class), .SDcols = -c(2:5)]

##          n   depth   table   price       x       y       z
## 1: integer numeric numeric integer numeric numeric numeric
```

# Contents

# Data Summarization

Goal: Summarize the characters of many variables of big data

- Amount of Information vs. Readability
- Various Variable Types vs. General Method/Representation
- Elaboration vs. Quickness/Easiness

Our Solution: Use data.table to convert each variable into key statistics, and create a new summarized data.table

# Summarize Variables (1): Type and Size

- First identify the type of each variable

```
data[,lapply(.SD, class)]

##          n   carat       cut     color   clarity   depth   table   price
## 1: integer numeric character character character numeric numeric integer
##          x       y       z
## 1: numeric numeric numeric
```

- count NA values of each variable

```
x = data[,price]
nonNA = function(x){
  return(sum(!is.na(x)))
}
nonNA(data[,price])

## [1] 39708

data[,lapply(.SD, nonNA)] # count of non-NA values of variables

##        n carat   cut color clarity depth table price     x     y     z
## 1: 53940 53940 53940 53940   53940 53940 53940 39708 53940 53940 53940
```

# Summarize Variables (2): Numeric Statistics

- Convert numeric variable into numeric statistics

```
summary_num = function(x){
  if(class(x) == "character")
    return(NA)
  else
    x_trans = c(mean(x, na.rm = T), sd(x, na.rm = T), quantile(x, na.rm = T))
  return(x_trans)
}
summary_num(data[,price])
```

```
##                              0%       25%       50%       75%      100%
##  3928.360  3992.675   326.000   949.000  2397.000  5302.250 18818.000
```

```
data[,lapply(.SD, summary_num)]
```

```
##           n     carat cut color clarity     depth     table     price
## 1: 26970.50 0.7979397  NA    NA      NA 61.749405 57.457184  3928.360
## 2: 15571.28 0.4740112  NA    NA      NA  1.432621  2.234491  3992.675
## 3:     1.00 0.2000000  NA    NA      NA 43.000000 43.000000   326.000
## 4: 13485.75 0.4000000  NA    NA      NA 61.000000 56.000000   949.000
## 5: 26970.50 0.7000000  NA    NA      NA 61.800000 57.000000  2397.000
## 6: 40455.25 1.0400000  NA    NA      NA 62.500000 59.000000  5302.250
## 7: 53940.00 5.0100000  NA    NA      NA 79.000000 95.000000 18818.000
##            x         y          z
## 1:  5.731157  5.734526  3.5387338
## 2:  1.121761  1.142135  0.7056988
## 3:  0.000000  0.000000  0.0000000
## 4:  4.710000  4.720000  2.9100000
## 5:  5.700000  5.710000  3.5300000
## 6:  6.540000  6.540000  4.0400000
## 7: 10.740000 58.900000 31.8000000
```

# Summarize Variables (3): Frequency Table

- We create a function to get the most 5 frequent values

```
summary_value = function(x){
  freq = sort(table(x), decreasing = TRUE)
  return(names(freq)[1:5])
}
data[,lapply(.SD, summary_value)]
```

```
##    n carat       cut color clarity depth table price    x    y    z
## 1: 1   0.3     Ideal     G     SI1    62    56   605 4.37 4.34  2.7
## 2: 2  0.31   Premium     E     VS2  61.9    57   828 4.34 4.37 2.69
## 3: 3  1.01 Very Good     F     SI2  61.8    58   776 4.33 4.35 2.71
## 4: 4   0.7      Good     H     VS1  62.2    59   789 4.38 4.33 2.68
## 5: 5  0.32      Fair     D    VVS2  62.1    55   666 4.32 4.32 2.72
```

- Get the frequency of most 5 frequent values, tail and NA

```
summary_freq = function(x){
  freq = sort(table(x), decreasing = TRUE)
  return(c(freq[1:5], sum(freq[-(1:5)]), sum(is.na(x))))
}
data[,lapply(.SD, summary_freq)]
```

```
##        n carat   cut color clarity depth table price     x     y     z
## 1:     1  2604 21551 11292   13065  2239  9881   103   448   437   767
## 2:     1  2249 13791  9797   12258  2163  9724    96   437   435   748
## 3:     1  2242 12082  9542    9194  2077  8369    95   429   425   738
## 4:     1  1981  4906  8304    8171  2039  6572    91   428   421   730
## 5:     1  1840  1610  6775    5066  2020  6268    88   425   414   697
## 6: 53935 43024     0  8230    6186 43402 13126 39235 51773 51808 50260
## 7:     0     0     0     0       0     0     0 14232     0     0     0
```

# Summarize Variables (4): Merge

- Use a list to collect these summaries, each reorganized by variable names as index

```
fL = list(class, nonNA, summary_num, summary_value, summary_freq)
summaryL = list()
for(i in 1:length(fL)){
  summaryL[[i]] = data[,lapply(.SD, fL[[i]])]
  summaryL[[i]] = data.table(t(summaryL[[i]]), keep.rownames=TRUE)
  setkey(summaryL[[i]], rn)
}
```

- Merge a list of data.tables into one data.table with Reduce()

```
summary_data = Reduce(function(X,Y){X[Y]}, summaryL)
colnames(summary_data) = c("variable", "type", "N",
                           "mean", "sd", "min", "Q1", "median", "Q3", "max",
                           paste("value", 1:5), paste("freq", c(1:5, "others", "NA")))
write.csv(summary_data, "Rlecture_Diamonds_summary.csv", row.names = FALSE, na = "")
```

# Contents

# Why Data Cleaning

- A key step to prepare big data for machine learning prediction
- Standardize raw data for statistical models to understand:
  - For programming generalization: convert data to numeric matrices for general-model use (beyond linear models in R)
  - Solve mathematical problems in data: sparsity, outliers, ...
- Difficulty: the size and complexity of variables in data

```
# data_big = fread("Rlecture_loan.csv")
# str(data_big)
# summary(data_big)
```

| Variable Problem | Example | Model Effect | Test |
|---|---|---|---|
| Non-numeric | characters, time | Error, Info loss | Data Type |
| Sparsity | Identical values | Noise | Count Frequency |
| Collinearity | Two similar variables | Instability | Correlation Matrix |
| NA values | NA, NA notations | Error | Count NAs |
| Distribution Bias | exponential, outliers | Instability | Numeric statistics |

# Data Cleaning (1): Numeralization

Our Goal: Convert all non-numeric variables into numeric varibles

- Non-numeric variables with prior information (ordinal, time, ...) $\rightarrow$ map into numeric values with a key-value data.table

```
mapDT = data.table(c("Fair", "Good", "Very Good", "Premium", "Ideal"), 1:5, key = "V1")
setkey(data, "cut")
op = data[mapDT]
data[, cut := op[,V2]]
```

- All other non-numeric variables $\rightarrow$ 0-1 dummy variables

```
options(na.action='na.pass')
data_clean = data.table(model.matrix( ~ . , data = data))
dim(data_clean)

## [1] 53940    23
```

# Data Cleaning (2): Detect and Delete Sparse Variables

- Check the number of NAs and most frequent values of each variable, define it as sparity

```
# data_clean[,lapply(.SD, summary_freq)]
sparsity = function(x){
  op = sort(table(x), decreasing = TRUE)[1] + sum(is.na(x))
  return(op/length(x))
}
(sparse = data_clean[ , lapply(.SD, sparsity)])
```

```
##    (Intercept)           n      carat       cut    colorE    colorF
## 1:           1 1.853912e-05 0.04827586 0.3995365 0.8183723 0.8230997
##       colorG    colorH    colorI    colorJ clarityIF claritySI1 claritySI2
## 1: 0.7906563 0.8460512 0.8994809 0.9479422  0.966815  0.7577864  0.8295514
##    clarityVS1 clarityVS2 clarityVVS1 clarityVVS2      depth    table
## 1:  0.8485169  0.7727475   0.9322395   0.9060808 0.04150908 0.183185
##        price           x           y         z
## 1: 0.2657582 0.008305525 0.008101594 0.0142195
```

- Delete all variables with sparsity nearly 1

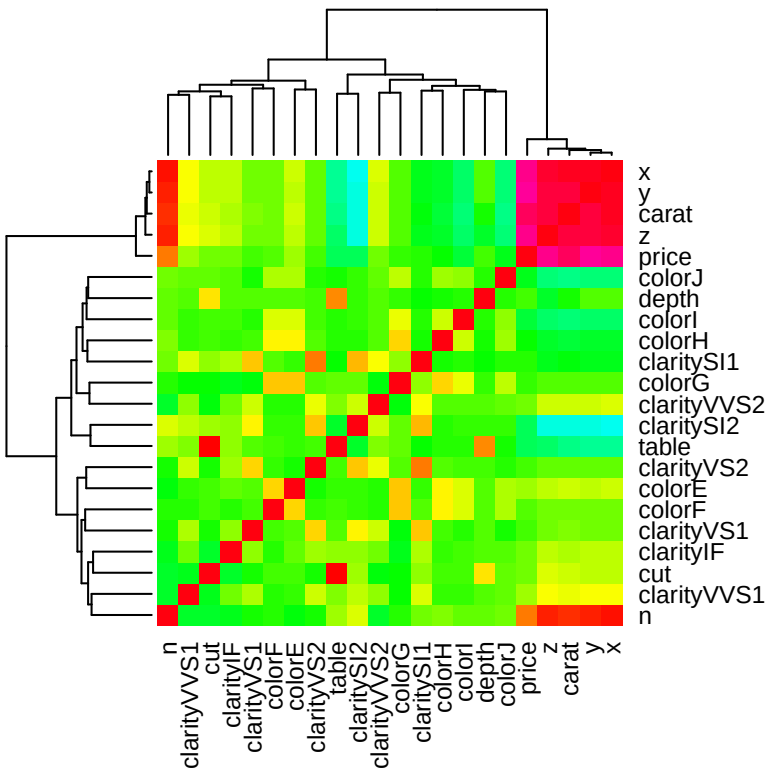
```
data_clean = data_clean[, data_clean[ ,sparse < 0.9999], with = FALSE]
dim(data_clean)
```

```
## [1] 53940    22
```

# Data Cleaning (3a): Detect Collinear Variables

- Use correlation matrix to detect collinearity

```
cm = cor(data_clean, use = "pairwise.complete.obs")
heatmap(cm, col = rainbow(100), scale = "none")
write.csv(cm, "Rlecture_Diamonds_cm.csv")
## Then open "Rlecture_Diamonds_cm.csv" in Microsoft Excel (2010+):
## Ctrl+A -> tag: Home -> Conditional Formatting -> Color Scales
```



# Data Cleaning (3b): Delete Collinear Variables

- Delete collinear variables with lower triangular of correlation matrix

```
(collinear = apply(lower.tri(cm) & (abs(cm) > 0.95), 1, sum))
```

```
##           n       carat         cut      colorE      colorF      colorG
##           0           0           0           0           0           0
##      colorH      colorI      colorJ   clarityIF  claritySI1  claritySI2
##           0           0           0           0           0           0
##  clarityVS1  clarityVS2 clarityVVS1 clarityVVS2       depth       table
##           0           0           0           0           0           0
##       price           x           y           z
##           0           1           2           3
```

```
data_clean = data_clean[, (names(collinear)[collinear == 0]), with = FALSE]
dim(data_clean)
```

```
## [1] 53940    19
```

# Data Cleaning (4): Fill NAs

- Check NA values

```
data_clean[, lapply(.SD, function(x){sum(is.na(x))})]

##    n carat cut colorE colorF colorG colorH colorI colorJ clarityIF
## 1: 0     0   0      0      0      0      0      0      0         0
##    claritySI1 claritySI2 clarityVS1 clarityVS2 clarityVVS1 clarityVVS2
## 1:          0          0          0          0           0           0
##    depth table price
## 1:     0     0 14232
```

- Fill NA values with the median/mean/0/... of the column (Not necessary for predicted variable Y)

```
fillNA = function(x){
  a = median(x, na.rm = TRUE)
  x[is.na(x) == TRUE] = a
  return(x)
}
data_clean = data_clean[, lapply(.SD, fillNA)]
```

# Data Cleaning (5): Distribution Normalization

- One method: Standardize all variables with mean 0 and standard deviation 1

```
data_clean = data_clean[, lapply(.SD, function(x){(x - mean(x))/sd(x)})]
```

- Another method: Standardize all variables to satisfy standard normal distributioon

```
data_clean = data_clean[, lapply(.SD, function(x){qnorm((frank(x)-0.5)/length(x))})]
```

# Contents

# Preparation

- Download the dataset "Rlecture_data_facebook.txt"

# Our Goals

- Learn network analysis in R with igraph
- Learn some basic concepts and methods to analyze a network
- Learn ERGM, a regression for network structure

# Contents

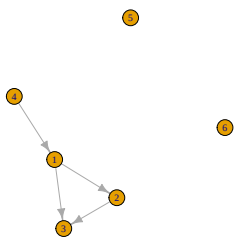
# Network Analysis

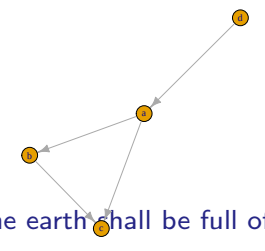
- There are many kinds of networks in the society: classmates, friends, telephones, business transactions,...
  - How they are same with and different from each other?
  - When we want to compare them, we need to represent them in a uniform way.
- Network (graph) representation in mathematics $G = (V, E)$
  - $G$ (Graph): the whole network
  - $V$ (Vertex): the nodes of network (people, companies, ...)
  - $E$ (Edges): the connection between nodes (friendship, transactions, ...)

# An R Network Analysis Package: igraph

- Install by install.packages("igraph")

```
library("igraph")
g = graph(edges = c(1,2, 2,3, 1,3, 4,1), n = 6)
plot(g)
g

## IGRAPH D--- 6 4 --
## + edges:
## [1] 1->2 2->3 1->3 4->1

V(g)

## + 6/6 vertices:
## [1] 1 2 3 4 5 6

E(g)

## + 4/4 edges:
## [1] 1->2 2->3 1->3 4->1

g_namev = graph(edges = c("a","b", "b","c", "a","c", "d","a"))
plot(g_namev)
```





# Directed and Undirected k-Stars

- We can also create simple graphs by graph.formula()

```
g_staru = graph.formula(B:C:D - A) # Undirected 3-Stars
plot(g_staru)
g_starb = graph.formula(B:C:D + A) # Bidirectional 3-Stars
plot(g_starb)
```

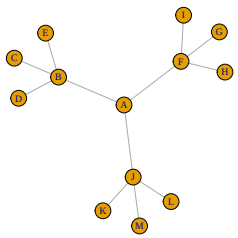

- Create one-sided directed graph with "+" on one side of "-"s

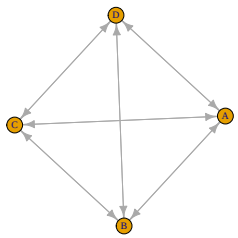
```
g_stari = graph.formula(B:C:D ---+ A) # In-directed 3-Stars
plot(g_stari)
g_staro = graph.formula(B:C:D +- A) # Out-directed 3-Stars
plot(g_staro)
```

# Some Other Basic Components of a Graph

- Tree graph and fully-connected graph

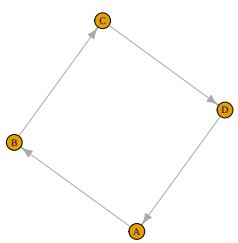
```
g_tree = graph.formula(A - B - C:D:E, A - F - G:H:I, A - J - K:M:L) # Tree Graph
plot(g_tree)
g_full = graph.formula(A:B:C:D + A:B:C:D) # Fully-connected Graph
plot(g_full)
```

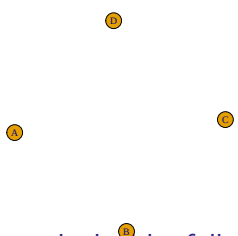




- Ring graph and isolated graph

```
g_ring = graph.formula(A -+ B -+ C -+ D -+ A) # Ring Graph
plot(g_ring)
g_empty = graph.formula(A:B:C:D) # Graph of Isolated Points
plot(g_empty)
```





# Demo Tutorial of Igraph

- Find and run the demo in igraph

```
demo(package = "igraph")
demo("centrality", package = "igraph")
```

- Use interactive version igraphdemo()

```
igraphdemo("centrality")
```

# Contents

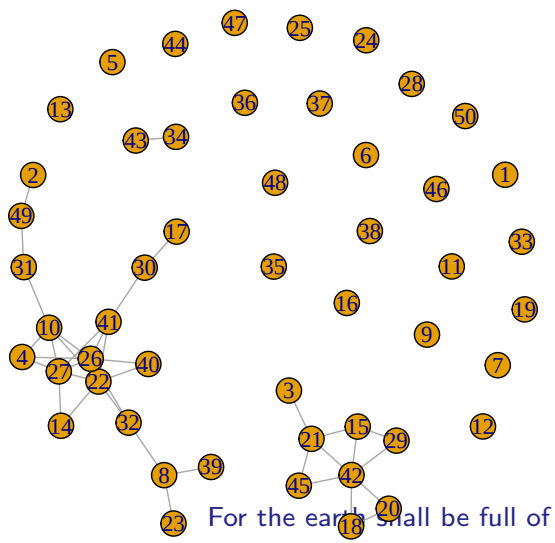
# Read Network Data

- Data: A sample of 4000+ Facebook friends (mutual)

```
datag = read.graph("Rlecture_data_facebook.txt", format = "edgelist")
datag = as.undirected(datag)
datag
```

- Take a subgraph from the original graph for analysis

```
datag_sub = induced.subgraph(datag, 1:50)
datag_sub
V(datag_sub)
E(datag_sub)
plot(datag_sub, vertex.size = 10, edge.arrow.size = 0.3)
```



# Neighbour and Path

- Neighbours: The set of vertices directly connected to a vertex

```
V(datag)[nei(1)] # The vertices vertex 1 is connected with

## + 2/4039 vertices:
## [1]  59 172
```

- The shortest path to pass from one vertex to another vertex

```
shortest.paths(datag, v = 4, to = 10) # The shortest path from vertex 4 to vertex 10

##      [,1]
## [1,]    1

shortest.paths(datag, v = 1:5, to = 1:10) # The shortest path from vertex 1~5 to vertex 1~10

##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## [1,]    0    3    7    4    5    4    7    4    7     3
## [2,]    3    0    6    2    4    2    6    2    6     2
## [3,]    7    6    0    5    7    6    4    5    4     5
## [4,]    4    2    5    0    5    2    6    3    5     1
## [5,]    5    4    7    5    0    4    7    3    7     4
```

# Adjacency Matrix

- Adjecency Matrix: the existence of edge between any two vertices

```
am = get.adjacency(datag)
am[1:10, 1:10]

## 10 x 10 sparse Matrix of class "dgCMatrix"
##
##  [1,] . . . . . . . . . .
##  [2,] . . . . . . . . . .
##  [3,] . . . . . . . . . .
##  [4,] . . . . . . . . . 1
##  [5,] . . . . . . . . . .
##  [6,] . . . . . . . . . .
##  [7,] . . . . . . . . . .
##  [8,] . . . . . . . . . .
##  [9,] . . . . . . . . . .
## [10,] . . . 1 . . . . . .
```

- Matrix power of adjecency matrix $A^p$: the total number of p-path between any two vertices

```
am2 = am%*%am
am2[1:10, 1:10]
```

# Measurements of Network Centrality (1)

- Degree $C_D(v)$: the number of edges of a vertex

```
vdegree = degree(datag)
head(vdegree) # The degree of first 6 vertices

## [1]  2 16  9 16  9 12

which.max(vdegree) # The vertex name with the highest degree

## [1] 2544

max(vdegree) # The highest degree of the network

## [1] 293
```

- Betweenness $C_B(v) = \sum_{i,j \in V \setminus \{v\}} \frac{\#\arg_v d(i,j)}{\#\arg d(i,j)}$

```
vbetween = betweenness(datag)
head(vbetween)

## [1]    0.000000 6293.454503    9.718594 3878.456505 7836.000000 4845.789988

which.max(vbetween)

## [1] 1086

max(vbetween)

## [1] 1951224
```

# Measurements of Network Centrality (2)

- Closeness $C(v) = \sum_{i \in V \setminus \{v\}} \frac{1}{d(i,v)}$

```
vclose = closeness(datag)
head(vclose)

## [1] 2.119906e-06 2.111067e-06 2.046526e-06 2.100739e-06 2.082804e-06
## [6] 2.101560e-06

which.max(vclose)

## [1] 1535

max(vclose)

## [1] 2.142516e-06
```

- Page Rank in Google

```
vpagerank = page.rank(datag)
head(vpagerank$vector)

## [1] 0.0000872622 0.0002273543 0.0002311486 0.0002231619 0.0002725550
## [6] 0.0002066856

which.max(vpagerank$vector)

## [1] 484

max(vpagerank$vector)

## [1] 0.001355442
```

# Comparison between Network Centrality

- We compare 4 measurements of network centrality

```
library(data.table)
vcentral = data.table(vertex = V(datag), degree = vdegree, betweenness = vbetween,
           closeness = vclose, pagerank = vpagerank$vector) # data.table of vertex centrality
setkey(vcentral, degree) # Sort by degree centrality
vcentral[,lapply(.SD, frank), .SDcols = -1] # Comparsion by rank
```

```
##       degree betweenness closeness pagerank
##    1:   40.5       174.5      40.5     40.5
##    2:   40.5       174.5      40.5     40.5
##    3:   40.5       174.5      40.5     40.5
##    4:   40.5       174.5      40.5     40.5
##    5:   40.5       174.5      40.5     40.5
##   ---
## 4035: 4035.0      3860.0    4025.0   4000.0
## 4036: 4036.0      3608.0    3980.0   4015.0
## 4037: 4037.0      3605.0    3961.0   4017.0
## 4038: 4038.0      3992.0    3419.0   3980.0
## 4039: 4039.0      4023.0    3844.0   4002.0
```

```
cor(vcentral[, -1, with = FALSE], method = "spearman") # Correlation matrix of rank (Spearman)
```

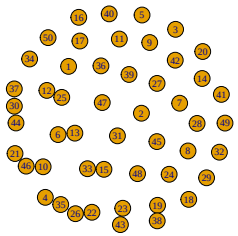
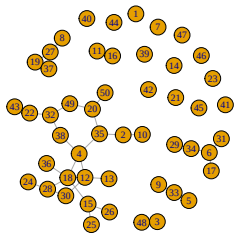
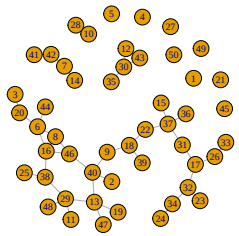
```
##                degree betweenness closeness  pagerank
## degree      1.0000000   0.5105299 0.6119082 0.7775559
## betweenness 0.5105299   1.0000000 0.4999893 0.6345377
## closeness   0.6119082   0.4999893 1.0000000 0.3113833
## pagerank    0.7775559   0.6345377 0.3113833 1.0000000
```

# Contents

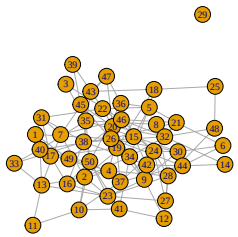
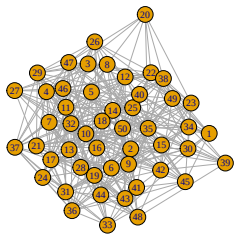
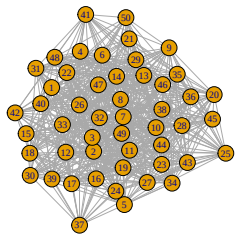
# Random Graph

- Random graph: edges randomly generated with probability p

```
set.seed(1)
plot(random.graph.game(50, p = 0.01))
plot(random.graph.game(50, p = 0.03))
plot(random.graph.game(50, p = 0.05))
```

```
plot(random.graph.game(50, p = 0.1))
plot(random.graph.game(50, p = 0.3))
plot(random.graph.game(50, p = 0.5))
```

# Exponential Random Graph Model (ERGM)

- Generate a new random graph based on shapes of network
- Assumption: the structure of a network $y$ can be represent as a exponential random graph with the number of shapes $s(y) = (s_1(y), ..., s_k(y))$ as statistics and $\beta = (\beta_1, ..., \beta_k)$ as coefficient
- Let $\beta = \hat{\beta}$ maximize
  $P(Y = y|\beta) = \exp\left\{\frac{1}{c}\sum_{j=1}^{k}\beta_j s_j(y) - c_1(\beta) - c_2(y)\right\}$
- Linear regression: Let $\beta = \hat{\beta}$ maximize
  $f(Y = y|\beta, X) = \exp\left\{\frac{1}{\sigma^2}\sum_{j=1}^{p}\beta_j(x_j^T y) - c_1(\beta, X) - c_2(y)\right\}$

# ERGM and Network Package

- Install by install.packages("ergm")

```
library(ergm)
data_edgelist = get.edgelist(datag)
datan = network(data_edgelist, directed = FALSE)
class(datag)

## [1] "igraph"

class(datan)

## [1] "network"
```

- We calculate the probability coefficient of edges for network

```
model_ergm1 = ergm(datan ~ edges, estimate = "MPLE")

## Evaluating log-likelihood at the estimate.

model_ergm1

##
## MPLE Coefficients:
##  edges
## -4.562
```

# Summary of ERGM

- All coefficients in ERGM represents an exponential magnitude in probability

```
summary(model_ergm1)

##
## ==========================
## Summary of model fit
## ==========================
##
## Formula:   datan ~ edges
##
## Iterations:  NA
##
## Maximum Likelihood Results:
##        Estimate Std. Error MCMC % p-value
## edges -4.562265   0.003463      0  <1e-04 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## For this model, the pseudolikelihood is the same as the likelihood.
##
##      Null Deviance: 11304871  on 8154741  degrees of freedom
##  Residual Deviance:   938040  on 8154740  degrees of freedom
##
## AIC: 938042    BIC: 938056    (Smaller is better.)
```

# Terms of ERGM (1)

To test the effect of a complex term, we must add all simpler terms to the model

- k-stars represent the concentration of relationships

```
model_ergm2 = ergm(datan ~ edges + kstar(2:3), estimate = "MPLE")

## Evaluating log-likelihood at the estimate.

summary(model_ergm2)

##
## ==========================
## Summary of model fit
## ==========================
##
## Formula:   datan ~ edges + kstar(2:3)
##
## Iterations:  NA
##
## Maximum Pseudolikelihood Results:
##          Estimate Std. Error MCMC % p-value
## edges  -7.534e+00  1.223e-02      0  <1e-04 ***
## kstar2  3.334e-02  1.467e-04      0  <1e-04 ***
## kstar3 -1.822e-04  1.402e-06      0  <1e-04 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Warning:  The standard errors are based on naive pseudolikelihood and are suspect.
##
##      Null Pseudo-deviance: 11304871  on 8154741  degrees of freedom
##  Residual Pseudo-deviance:   779779  on 8154738  degrees of freedom
```

# Terms of ERGM (2)

- Triangles represent a smallest clique in the network

```
model_ergm3 = ergm(datan ~ edges + kstar(2:3) + triangle, estimate = "MPLE")

## Evaluating log-likelihood at the estimate.

summary(model_ergm3)

##
## ==========================
## Summary of model fit
## ==========================
##
## Formula:   datan ~ edges + kstar(2:3) + triangle
##
## Iterations:  NA
##
## Maximum Pseudolikelihood Results:
##             Estimate Std. Error MCMC % p-value
## edges     -5.790e+00  1.199e-02      0  <1e-04 ***
## kstar2     1.537e-02  2.192e-04      0  <1e-04 ***
## kstar3    -4.302e-04  2.712e-06      0  <1e-04 ***
## triangle   1.624e-01  5.043e-04      0  <1e-04 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Warning:  The standard errors are based on naive pseudolikelihood and are suspect.
##
##      Null Pseudo-deviance: 11304871  on 8154741  degrees of freedom
##  Residual Pseudo-deviance:   437289  on 8154737  degrees of freedom
##
## AIC: 437297    BIC: 437353    (Smaller is better.)
```

# Terms of ERGM (3): More Terms

We can also add more terms to ERGM

- concurrent: vertices with degree 2 or more

```
model_ergm4 = ergm(datan ~ edges + concurrent, estimate = "MPLE")
summary(model_ergm4)
```

- threetrail: a path of 3 connected edges

```
model_ergm5 = ergm(datan ~ edges + kstar(2) + threetrail, estimate = "MPLE")
summary(model_ergm5)
```

- cycle(k): a ring of k edges. In an undirected graph, cycle(3) are triangles, and cycle(4) are squares

```
model_ergm6 = ergm(datan ~ edges + kstar(2:3) + threetrail + cycle(3:4), estimate = "MPLE")
summary(model_ergm5)
```

# Contents

# Preparation

- Download the dataset "RIecture_Diamonds.csv"

# Our Goals

- Review the contents of first 4 lecture that we learned
- Be familiar with package "data.table" and "ggplot2" in R data analysis

# Contents

# Vectors

- Create vectors with different methods

```
x = seq(0, 3, length.out = 6)
x

## [1] 0.0 0.6 1.2 1.8 2.4 3.0

y = rep(c("a", "b"), 3)
y

## [1] "a" "b" "a" "b" "a" "b"

set.seed(1)
z = runif(6, min = 0, max = 1) # Distribution function (normal)
z

## [1] 0.2655087 0.3721239 0.5728534 0.9082078 0.2016819 0.8983897
```

- Select/revise elements of vectors by position and logical index

```
x[c(1, 3, 5)]

## [1] 0.0 1.2 2.4

x[y == "a"] = 1
x

## [1] 1.0 0.6 1.0 1.8 1.0 3.0
```

# Vector Functions

- Use vector functions to operate on vectors

```
x ^ 2

## [1] 1.00 0.36 1.00 3.24 1.00 9.00

pnorm(x, mean = 0, sd = 1)

## [1] 0.8413447 0.7257469 0.8413447 0.9640697 0.8413447 0.9986501

sum(x)

## [1] 8.4
```

- Use paste function to operate on characters

```
y1 = paste(y, x, sep = "_")
y1

## [1] "a_1"   "b_0.6" "a_1"   "b_1.8" "a_1"   "b_3"

substr(y1, start = 1, stop = 3)

## [1] "a_1" "b_0" "a_1" "b_1" "a_1" "b_3"
```

# Matrix

- Construct a matrix by a vector

```
m = matrix(c(x, z), nrow = 4)
m
```

```
##      [,1]      [,2]      [,3]
## [1,]  1.0 1.0000000 0.5728534
## [2,]  0.6 3.0000000 0.9082078
## [3,]  1.0 0.2655087 0.2016819
## [4,]  1.8 0.3721239 0.8983897
```

- Select/revise elements of a matrix

```
m[, 2]
```

```
## [1] 1.0000000 3.0000000 0.2655087 0.3721239
```

```
m[1:3, 2:3] = 5
m
```

```
##      [,1]      [,2]      [,3]
## [1,]  1.0 5.0000000 5.0000000
## [2,]  0.6 5.0000000 5.0000000
## [3,]  1.0 5.0000000 5.0000000
## [4,]  1.8 0.3721239 0.8983897
```

# Matrix Functions

- Matrix calculation

```
m2 = t(m) %*% m
dim(m2)

## [1] 3 3

det(m2)

## [1] 2.215645
```

- Apply vector functions on matrices by column/row

```
apply(m, 1, sum)

## [1] 11.000000 10.600000 11.000000  3.070514
```

- Bind matrices by column/row: cbind()/rbind()

```
cbind(m, m)

##      [,1]      [,2]      [,3] [,4]      [,5]      [,6]
## [1,]  1.0 5.0000000 5.0000000  1.0 5.0000000 5.0000000
## [2,]  0.6 5.0000000 5.0000000  0.6 5.0000000 5.0000000
## [3,]  1.0 5.0000000 5.0000000  1.0 5.0000000 5.0000000
## [4,]  1.8 0.3721239 0.8983897  1.8 0.3721239 0.8983897
```

# List

- Construct a list with key-value pairs

```
d = list(a = x, b = y, c = m2)
d

## $a
## [1] 1.0 0.6 1.0 1.8 1.0 3.0
##
## $b
## [1] "a" "b" "a" "b" "a" "b"
##
## $c
##          [,1]     [,2]     [,3]
## [1,]  5.60000 13.66982 14.61710
## [2,] 13.66982 75.13848 75.33431
## [3,] 14.61710 75.33431 75.80710
```

- Select a sublist and revise elements of a list

```
d[c("a", "b")]

## $a
## [1] 1.0 0.6 1.0 1.8 1.0 3.0
##
## $b
## [1] "a" "b" "a" "b" "a" "b"

d$c = z
d[["c"]]

## [1] 0.2655087 0.3721239 0.5728534 0.9082078 0.2016819 0.8983897
```

# List Functions

- lapply/sapply: use functions on each elements of a list

```
lapply(d, length)

## $a
## [1] 6
##
## $b
## [1] 6
##
## $c
## [1] 6

sapply(d, length)

## a b c
## 6 6 6
```

# Data Frame

- Create a data.frame from a list of equal-length vectors

```
data = data.frame(d)
data
```

```
##     a b         c
## 1 1.0 a 0.2655087
## 2 0.6 b 0.3721239
## 3 1.0 a 0.5728534
## 4 1.8 b 0.9082078
## 5 1.0 a 0.2016819
## 6 3.0 b 0.8983897
```

- We can select from a data.frame like matrix and list

```
data[4:5, ]
```

```
##     a b         c
## 4 1.8 b 0.9082078
## 5 1.0 a 0.2016819
```

```
data$a
```

```
## [1] 1.0 0.6 1.0 1.8 1.0 3.0
```

```
data[data$b == "a", c("b", "c")]
```

```
##   b         c
## 1 a 0.2655087
## 3 a 0.5728534
## 5 a 0.2016819
```

# Data Frame Functions

- Create a data.frame from a list of equal-length vectors

```
dim(data)

## [1] 6 3
```

- Use either apply for a matrix or lapply/sapply for a list

```
apply(data[,-2], 2, sum)

##        a        c
## 8.400000 3.218765

apply(data[,-2], 1, sum)

## [1] 1.2655087 0.9721239 1.5728534 2.7082078 1.2016819 3.8983897

sapply(data[,-2], sum)

##        a        c
## 8.400000 3.218765
```

# Contents

# Data Table: A Poweful Extension of Data Frame

- Read data

```
library("data.table")
data = fread("Rlecture_Diamonds.csv")
```

- Summarize with str() and summary()

```
str(data)
summary(data)
```

- Select rows and columns from a data.table

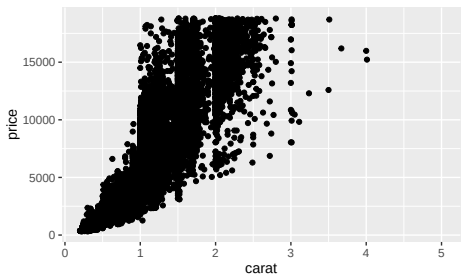
```
data[4:7, list(n, carat, price)]

##    n carat price
## 1: 4  0.29   334
## 2: 5  0.31   335
## 3: 6  0.24    NA
## 4: 7  0.24    NA

data[rank(price) <= 5, list(n, carat, price, unit_price = price/carat)]

##    n carat price unit_price
## 1: 2  0.21   326   1552.381
## 2: 4  0.29   334   1151.724
## 3: 5  0.31   335   1080.645
## 4: 8  0.26   337   1296.154
## 5: 9  0.22   337   1531.818
```

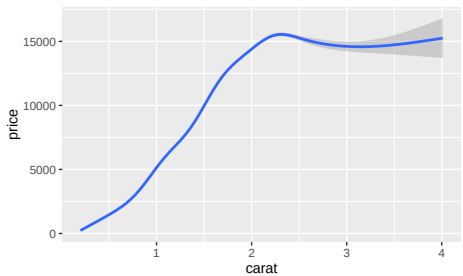
# ggplot2: An Advanced Plot for Data Frame

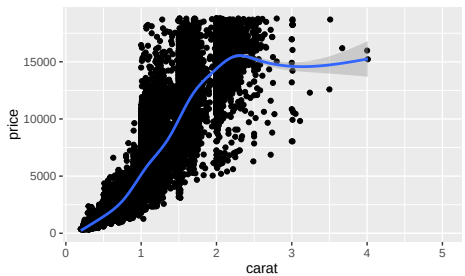
ggplot(data, mapping = aes(x = x, y = y, ...)) + geom...()

- ggplot(data, mapping): prepare data for plot
- aes(): select variables to map (x,y) and groupby (fill/color/...)
- geom...(): how we show the data we select in ggplot

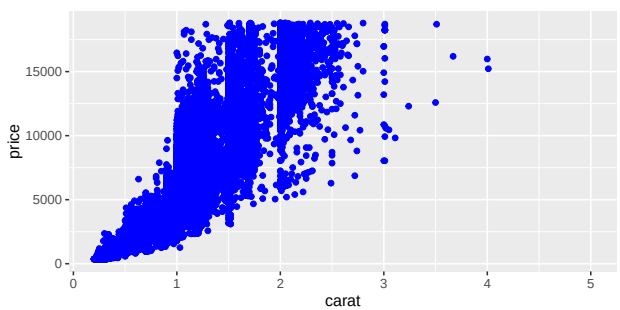
```
library("ggplot2")
ggplot(data, aes(x = carat, y = price)) + geom_point()
ggplot(data, aes(x = carat, y = price)) + geom_smooth()
ggplot(data, aes(x = carat, y = price)) + geom_point() + geom_smooth()
```

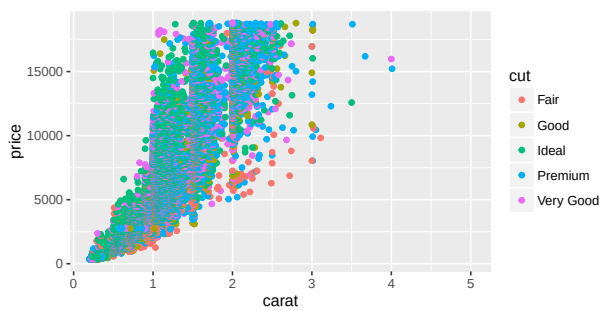



```
ggplot(data, aes(x = carat, y = price)) + geom_point(color = "blue")
ggplot(data, aes(x = carat, y = price, color = cut)) + geom_point()
```



# Frequency and Distribution

- Plot: bar→categorical and histogram/density→numeric

```
ggplot(data, aes(x = cut)) + geom_bar(aes(fill = color))
ggplot(data, aes(x = cut)) + geom_bar(aes(fill = price > 1000), position=position_dodge())
ggplot(data, aes(x = carat)) + geom_histogram(aes(fill = cut), binwidth = 0.1)
ggplot(data, aes(x = carat)) + geom_density(aes(fill = cut), alpha = 0.5)
```


- Counting in data.table, use .N (the length) and by = ...

```
data[, list(count = .N), by = cut]

##          cut count
## 1:     Ideal 21551
## 2:   Premium 13791
## 3:      Good  4906
## 4: Very Good 12082
## 5:      Fair  1610

data[, list(count = .N), by = list(carat = cut(carat, breaks = 3))]

##          carat count
## 1: (0.195,1.8] 51666
## 2:  (1.8,3.41]  2264
## 3: (3.41,5.01]    10
```

# Groupby in Data Table

- data.table has the form: data[i, j, by = ...]

```
data[i = price > 1000, j = list(count = .N, mean = mean(carat), sd = sd(carat))]

##    count      mean        sd
## 1: 29024 0.9679445 0.4457943

data[i = price > 1000, j = list(count = .N, mean = mean(carat), sd = sd(carat)), by = list(cut)]

##           cut count      mean        sd
## 1:      Fair  1114 1.0813914 0.4774036
## 2:     Ideal 10814 0.8757287 0.4268839
## 3: Very Good  6507 0.9784555 0.4122846
## 4:      Good  2795 0.9967263 0.4145200
## 5:   Premium  7794 1.0605799 0.4781261
```

- Groupby functions in data.table, ggplot2 and SQL (a database language)

| R (data.table) | i | j | by |
|---|---|---|---|
| R (ggplot2) | | aes(x, y) | aes(color/fill/...) |
| SQL | WHERE | SELECT | GROUP BY |

# Column Apply in Data Table

- Apply on each column in data.table with lapply(.SD, function)

```
data[,lapply(.SD, class)]

##          n   carat       cut     color   clarity   depth   table   price
## 1: integer numeric character character character numeric numeric integer
##          x       y       z
## 1: numeric numeric numeric
```

- We can also define functions for use

```
summary_num = function(x){
  if(class(x) == "character") return(NA)
  else return(c(mean = mean(x, na.rm = T), sd = sd(x, na.rm = T), quantile(x, na.rm = T)))
}
summary_num(data[,price])

##      mean        sd        0%       25%       50%       75%      100%
##  3928.360  3992.675   326.000   949.000  2397.000  5302.250 18818.000

data[,lapply(.SD, summary_num)]

##           n     carat cut color clarity     depth     table     price
## 1: 26970.50 0.7979397  NA    NA      NA 61.749405 57.457184  3928.360
## 2: 15571.28 0.4740112  NA    NA      NA  1.432621  2.234491  3992.675
## 3:     1.00 0.2000000  NA    NA      NA 43.000000 43.000000   326.000
## 4: 13485.75 0.4000000  NA    NA      NA 61.000000 56.000000   949.000
## 5: 26970.50 0.7000000  NA    NA      NA 61.800000 57.000000  2397.000
## 6: 40455.25 1.0400000  NA    NA      NA 62.500000 59.000000  5302.250
## 7: 53940.00 5.0100000  NA    NA      NA 79.000000 95.000000 18818.000
##            x         y          z
## 1:  5.731157  5.734526  3.5387338
## 2:  1.121761  1.142135  0.7056988
```

# Contents

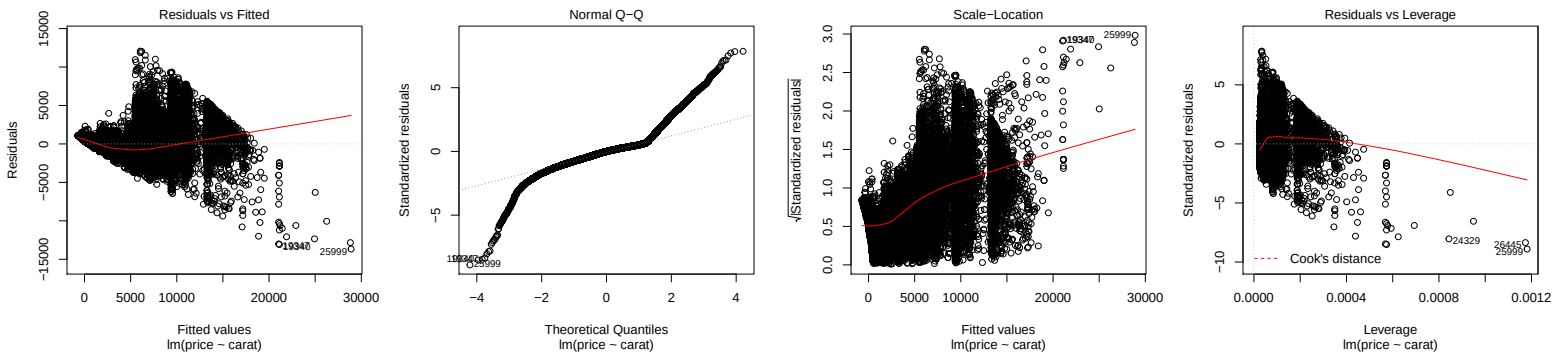
# Linear Model and Residuals

- Build a linear model in R with lm(formula = ..., data)

```
options(na.action = 'na.exclude')
model = lm(price ~ carat, data = data)
summary(model)
```

- Use plot for residual analysis

```
plot(model)
```



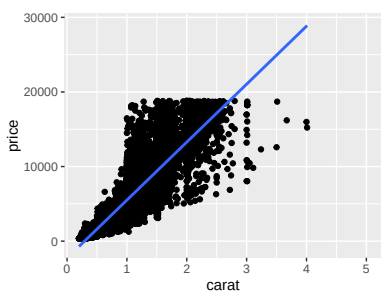
# Variable Transformation

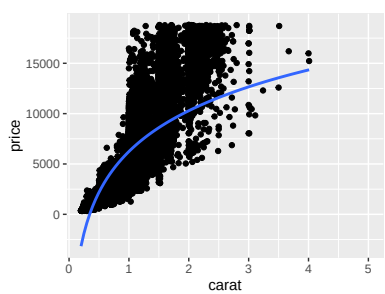
- We can use transformation functions on x and y

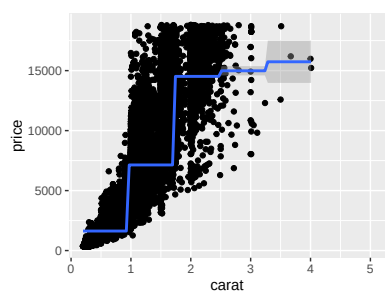
```
model_log = lm(price ~ log(carat), data = data)
summary(model_log)
model_cut = lm(price ~ cut(carat, 5), data = data)
summary(model_cut)
model_poly1 = lm(price ~ poly(carat, 3), data = data)
summary(model_poly1)
model_poly2 = lm(price ~ poly(carat, 3, raw = TRUE), data = data)
summary(model_poly2)
```

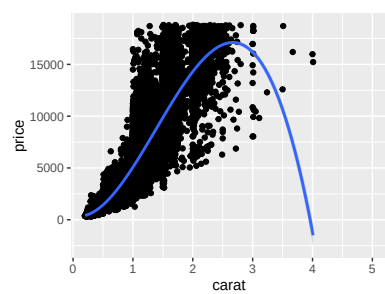
- Use ggplot to plot univariate linear models

```
ggplot(data, aes(x=carat,y=price)) + geom_point() + geom_smooth(method="lm")
ggplot(data, aes(x=carat,y=price)) + geom_point() + geom_smooth(method="lm", formula=y~log(x))
ggplot(data, aes(x=carat,y=price)) + geom_point() + geom_smooth(method="lm", formula=y~cut(x,5))
ggplot(data, aes(x=carat,y=price)) + geom_point() + geom_smooth(method="lm", formula=y~poly(x,3))
```









# Multivariate Linear Models and ANOVA

- Use "+": adding variables, ":": interaction, and "*": for both

```
model1 = lm(price ~ carat + cut + carat:cut, data = data)
summary(model1)
model2 = lm(price ~ carat*cut, data = data)
summary(model2)
```

- Use ANOVA to test the effect of multi variables in the linear model

```
anova(model1)
```

```
## Analysis of Variance Table
##
## Response: price
##               Df     Sum Sq    Mean Sq   F value    Pr(>F)
## carat          1 5.3925e+11 5.3925e+11 243069.75 < 2.2e-16 ***
## cut            4 4.3910e+09 1.0977e+09    494.82 < 2.2e-16 ***
## carat:cut      4 1.2771e+09 3.1927e+08    143.91 < 2.2e-16 ***
## Residuals 39698 8.8070e+10 2.2185e+06
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

# High-Dimensional Variables and Selection

- Put all variables except n into the model, ".": all other variables, "-": remove a variable

```
model_all = lm(price ~ . - n, data = data)
summary(model_all)
```

- Stepwise selection with both forward and backward method

```
model_step = step(model_all, direction = "both", trace = 0)
anova(model_step)

## Analysis of Variance Table
##
## Response: price
##               Df     Sum Sq    Mean Sq    F value    Pr(>F)
## carat          1 5.3925e+11 5.3925e+11 4.3204e+05 < 2.2e-16 ***
## cut            4 4.3910e+09 1.0977e+09 8.7951e+02 < 2.2e-16 ***
## color          6 9.2122e+09 1.5354e+09 1.2301e+03 < 2.2e-16 ***
## clarity        7 2.7850e+10 3.9786e+09 3.1877e+03 < 2.2e-16 ***
## depth          1 1.6634e+06 1.6634e+06 1.3327e+00    0.2483
## table          1 6.6218e+07 6.6218e+07 5.3054e+01 3.306e-13 ***
## x              1 2.6829e+09 2.6829e+09 2.1495e+03 < 2.2e-16 ***
## Residuals 39686 4.9534e+10 1.2481e+06
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```