An R Lecture from Practice: Part I

Fangda Fan

2016.5

Contents

Preparation

- Bring your laptop.
- Install R **and** R-studio
- Download the dataset "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?

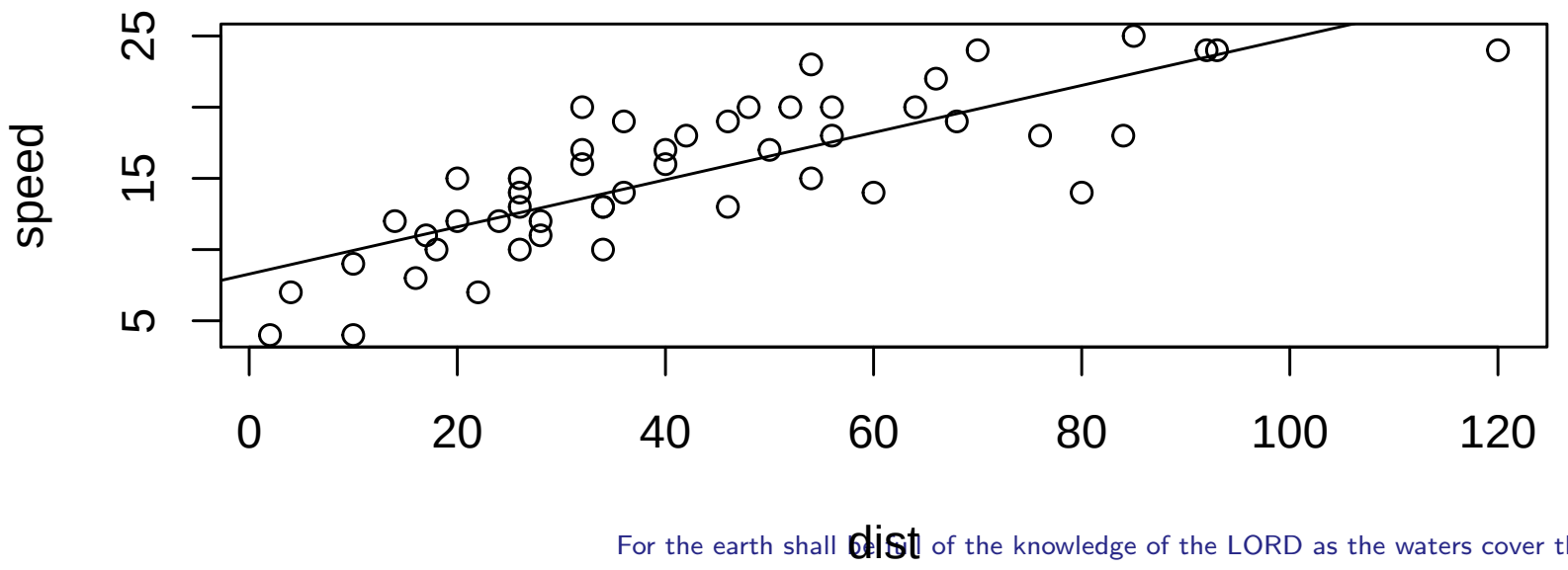
- Data: Input, output, cleaning, extracting, summarizing,...
- Statistical models: random variables and distribution, linear regression, clustering, machine learning, social analysis, ..., nearly all statistical methods that you can think out of
- Scientific computing: vector and matrix operations, lots 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 "Diamonds.csv"

Our Goals

- Read real data from files
- Analyze data with R Data Frame

Data

- Information of over 50000 diamonds
- Analyze data with R Data Frame

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 "$\cdots$" 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("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
```

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()

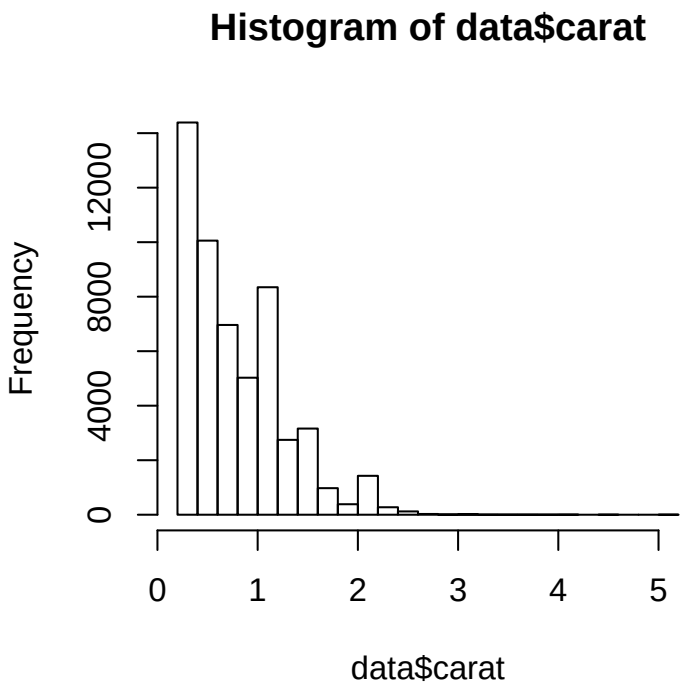
```
aggregate(carat ~ cut, data, mean) # summarize by groups

##         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
```

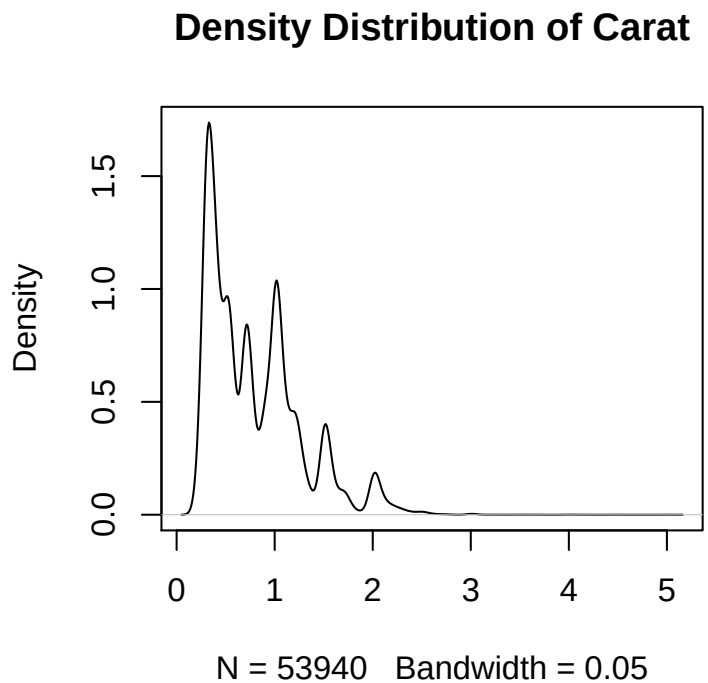
Contents

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")
```

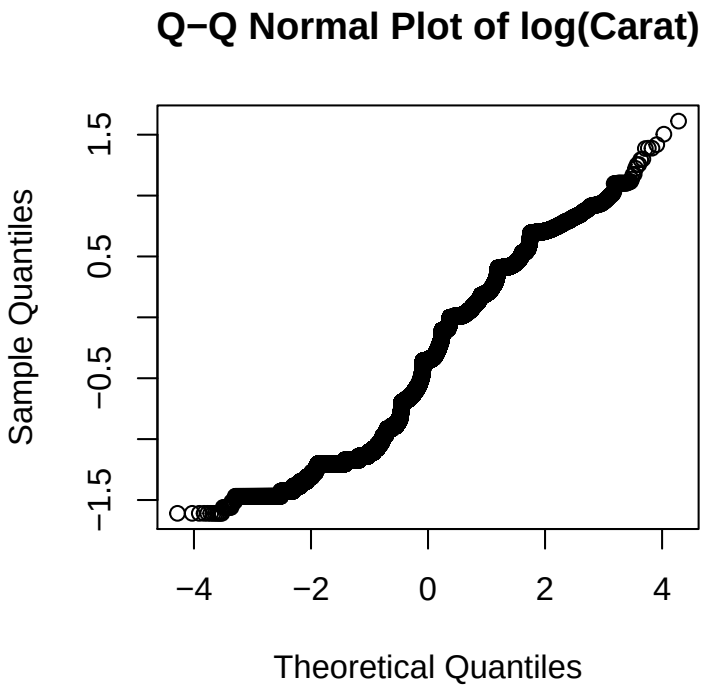


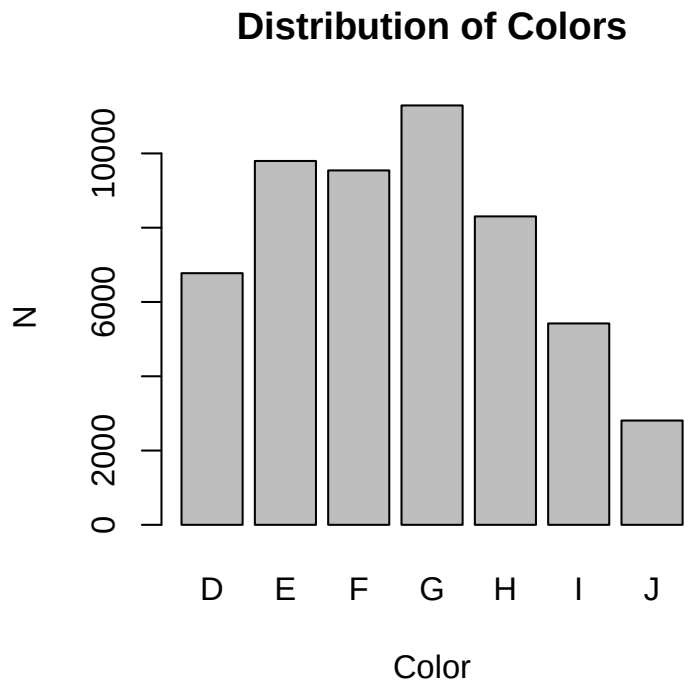


Univariate Plot (2)

- 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")
```

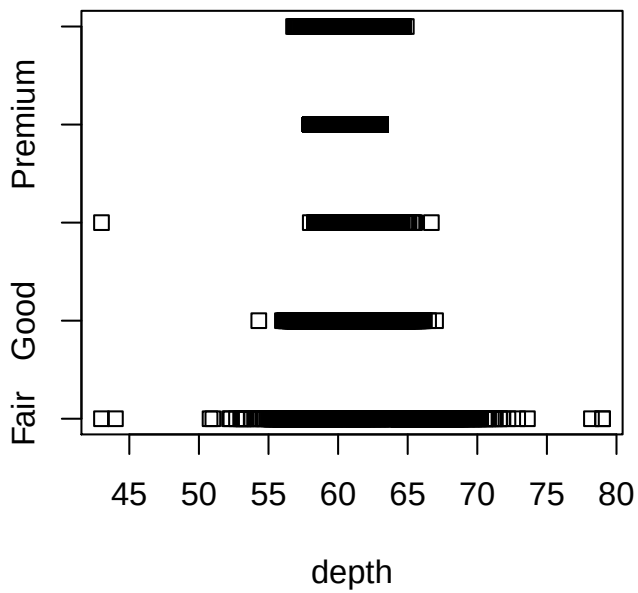


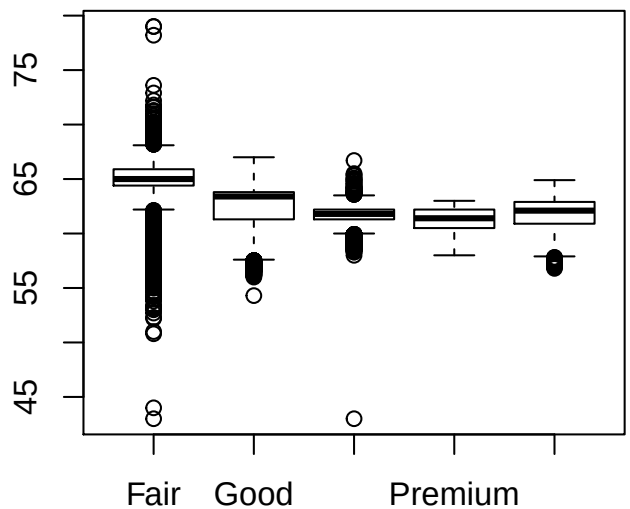


Bivariate Plot (1)

- 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
```

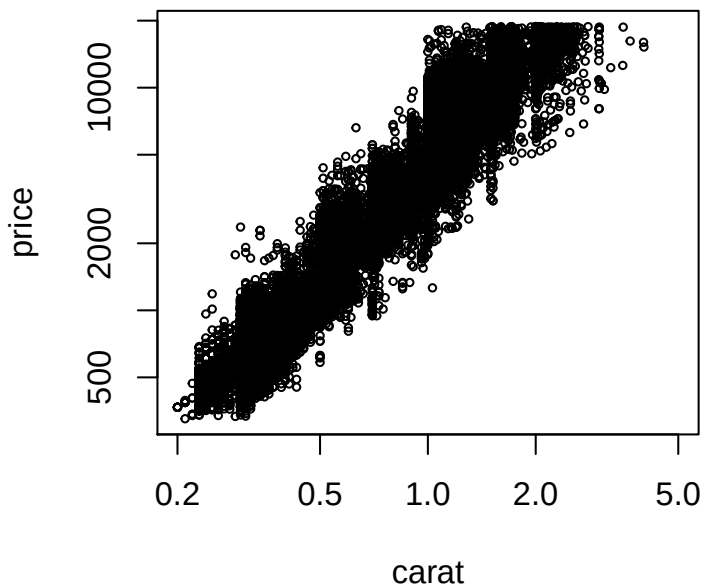


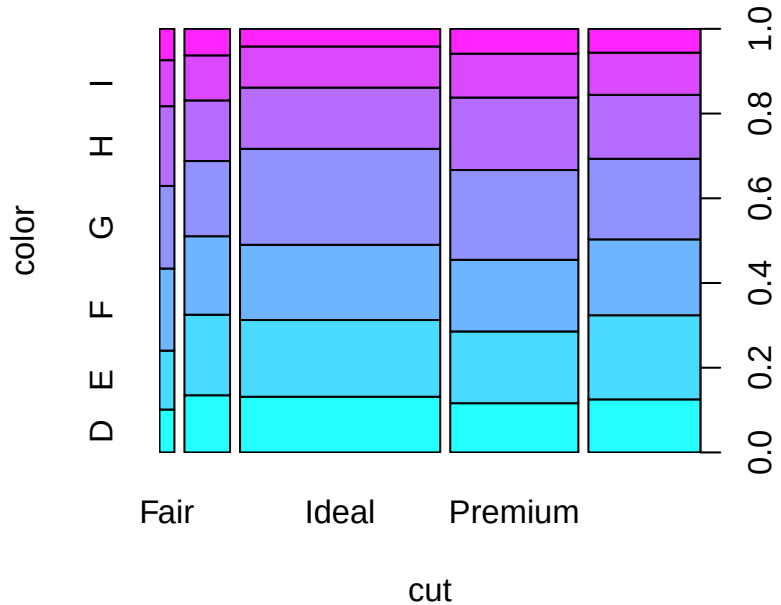


Bivariate Plot (2)

- 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

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
```

Select a subset from data

- Select where carat $>$ 1 and column (carat, color, price) from data by subset()

```
data_sub_1 = subset(data, carat > 1, select = c(carat, color, price))
str(data_sub_1)

## 'data.frame':^^I17502 obs. of  3 variables:
##  $ carat: num  1.17 1.01 1.01 1.01 1.05 1.05 1.01 1.04 1.2 1.02 ...
##  $ color: Factor w/ 7 levels "D","E","F","G",..: 7 3 2 5 7 7 2 4 3 4 ...
##  $ price: num  2774 2781 2788 2788 2397 ...
```

- Select the same by row and column of data frame

```
data_sub_2 = data[data$carat > 1, c("carat", "color", "price")]
str(data_sub_2)

## 'data.frame':^^I17502 obs. of  3 variables:
##  $ carat: num  1.17 1.01 1.01 1.01 1.05 1.05 1.01 1.04 1.2 1.02 ...
##  $ color: Factor w/ 7 levels "D","E","F","G",..: 7 3 2 5 7 7 2 4 3 4 ...
##  $ price: num  2774 2781 2788 2788 2397 ...
```

Contents

Theory: Linear Regression

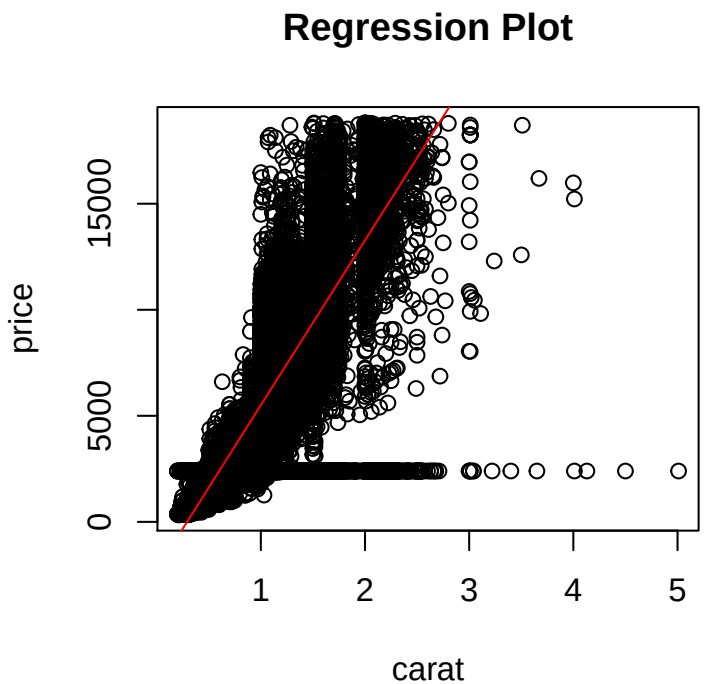
Model: $Y_{n\times 1} = \beta_0 + X_{n\times p}\beta_{p\times 1} + \varepsilon_{n\times 1}$

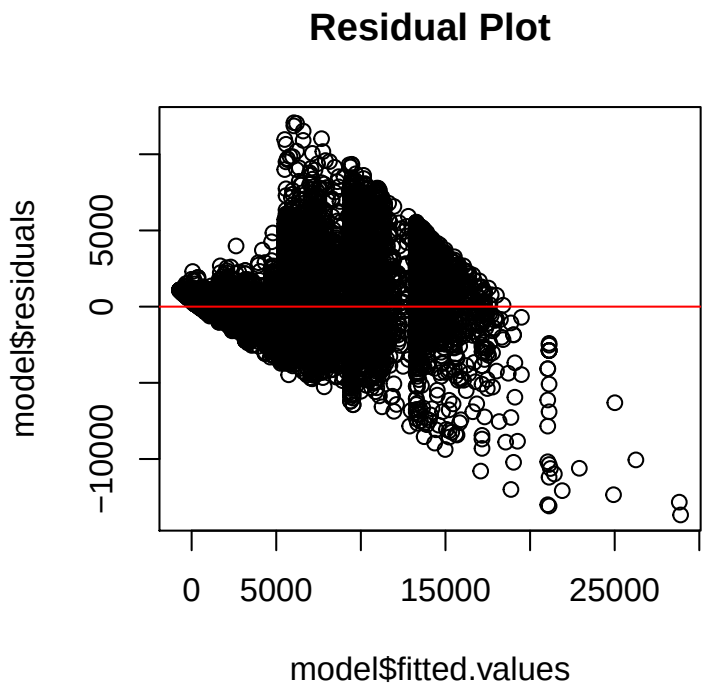
- $\varepsilon_{n\times 1} = (\varepsilon_1, \varepsilon_2, \cdots, \varepsilon_n)^T \sim^{iid} N(0, \sigma^2), \quad \sigma^2 > 0$
- Given $Y_{n\times 1} = (y_1, y_2, \cdots, y_n)^T$ and $X_{n\times p} = (X_1, X_2, \cdots, X_p)$
- Use least squares method to solve β_0 and $\beta_{p\times 1} = (\beta_1, \beta_2, \cdots, \beta_p)^T$

Single Variable Regression

- Let Y: price, X: carat

```
model = lm(price ~ carat, data = data_clean)
plot(price ~ carat, data = data, 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

- Let Y: price, X: carat, cut(factor variable) 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
```

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)
```