An R Lecture from Practice: Part III

Fangda Fan

2016.5

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

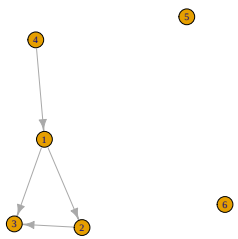
Network Analysis

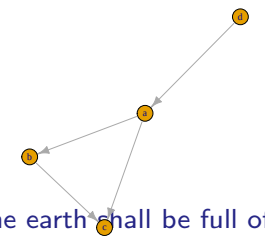
- There are many kinds of networks in the society: classmates, friends, telephones, business transactions,...
  - How they are same with and different from each other?
  - When we want to compare them, we need to represent them in a uniform way.
- Network (graph) representation in mathematics $G = (V,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

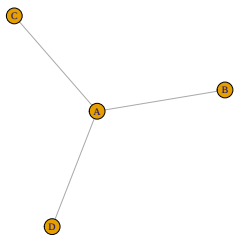
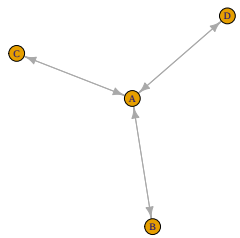




Directed and Undirected k-Stars

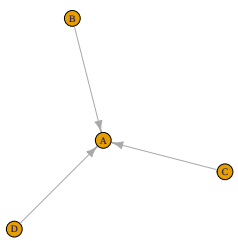
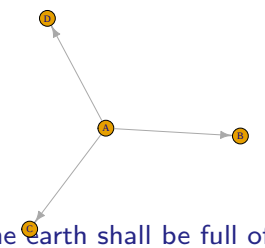
- We can also create simple graphs by graph.formula()

```
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

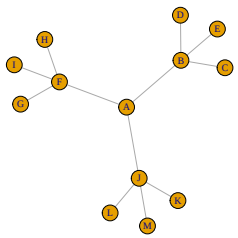
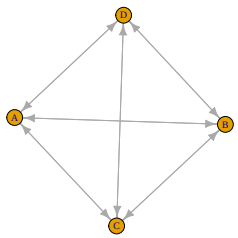
```
g_stari = graph.formula(B:C:D ---+ A) # In-directed 3-Stars
plot(g_stari)
g_staro = graph.formula(B:C:D +- A) # Out-directed 3-Stars
plot(g_staro)
```

Some Other Basic Components of a Graph

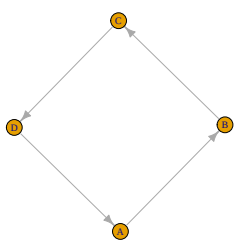
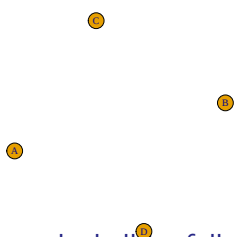
- Tree graph and fully-connected graph

```
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

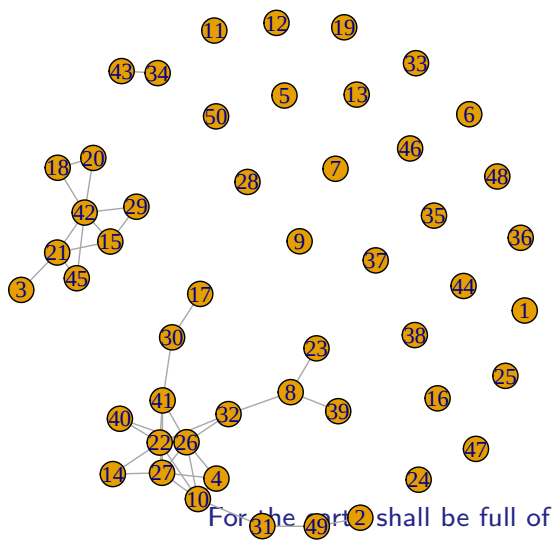
Read Network Data

- 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]    0.00 6293.45    9.72 3878.46 7836.00 4845.79

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] 2.12e-06 2.11e-06 2.05e-06 2.10e-06 2.08e-06 2.10e-06

which.max(vclose)

## [1] 1535

max(vclose)

## [1] 2.14e-06
```

- Page Rank in Google

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

## [1] 8.73e-05 2.27e-04 2.31e-04 2.23e-04 2.73e-04 2.07e-04

which.max(vpagerank$vector)

## [1] 484

max(vpagerank$vector)

## [1] 0.00136
```

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      40.5     40.5
##    2:   40.5         174      40.5     40.5
##    3:   40.5         174      40.5     40.5
##    4:   40.5         174      40.5     40.5
##    5:   40.5         174      40.5     40.5
##   ---
## 4035: 4035.0        3860    4025.0   4000.0
## 4036: 4036.0        3608    3980.0   4015.0
## 4037: 4037.0        3605    3961.0   4017.0
## 4038: 4038.0        3992    3419.0   3980.0
## 4039: 4039.0        4023    3844.0   4002.0
```

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

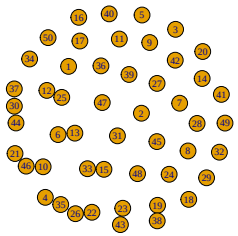
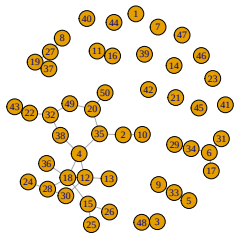
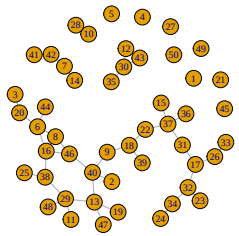
```
##             degree betweenness closeness pagerank
## degree       1.000       0.511     0.612    0.778
## betweenness  0.511       1.000     0.500    0.635
## closeness    0.612       0.500     1.000    0.311
## pagerank     0.778       0.635     0.311    1.000
```

Contents

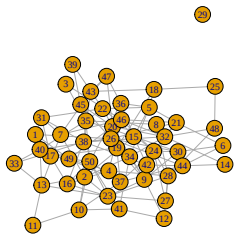
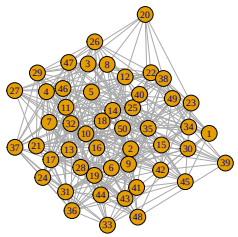
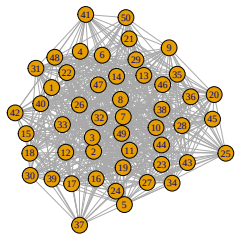
Random Graph

- 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
## -4.56
```

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 -4.56226    0.00346      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  -7.53e+00   1.22e-02      0  <1e-04 ***
## kstar2  3.33e-02   1.47e-04      0  <1e-04 ***
## kstar3 -1.82e-04   1.40e-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    -5.79e+00   1.20e-02      0  <1e-04 ***
## kstar2    1.54e-02   2.19e-04      0  <1e-04 ***
## kstar3   -4.30e-04   2.71e-06      0  <1e-04 ***
## triangle  1.62e-01   5.04e-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] 0.266 0.372 0.573 0.908 0.202 0.898
```

- 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] 0.841 0.726 0.841 0.964 0.841 0.999

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 1.000 0.573
## [2,]  0.6 3.000 0.908
## [3,]  1.0 0.266 0.202
## [4,]  1.8 0.372 0.898
```

- Select/revise elements of a matrix

```
m[, 2]
```

```
## [1] 1.000 3.000 0.266 0.372
```

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

```
##      [,1]  [,2]  [,3]
## [1,]  1.0 5.000 5.000
## [2,]  0.6 5.000 5.000
## [3,]  1.0 5.000 5.000
## [4,]  1.8 0.372 0.898
```

Matrix Functions

- Matrix calculation

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

## [1] 3 3

det(m2)

## [1] 2.22
```

- Apply vector functions on matrices by column/row

```
apply(m, 1, sum)

## [1] 11.00 10.60 11.00  3.07
```

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

```
cbind(m, m)

##      [,1]  [,2]  [,3] [,4]  [,5]  [,6]
## [1,]  1.0 5.000 5.000  1.0 5.000 5.000
## [2,]  0.6 5.000 5.000  0.6 5.000 5.000
## [3,]  1.0 5.000 5.000  1.0 5.000 5.000
## [4,]  1.8 0.372 0.898  1.8 0.372 0.898
```

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,]  5.6 13.7 14.6
## [2,] 13.7 75.1 75.3
## [3,] 14.6 75.3 75.8
```

- 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] 0.266 0.372 0.573 0.908 0.202 0.898
```

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 0.266
## 2 0.6 b 0.372
## 3 1.0 a 0.573
## 4 1.8 b 0.908
## 5 1.0 a 0.202
## 6 3.0 b 0.898
```

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

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

```
##     a b     c
## 4 1.8 b 0.908
## 5 1.0 a 0.202
```

```
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 0.266
## 3 a 0.573
## 5 a 0.202
```

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
## 8.40 3.22

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

## [1] 1.266 0.972 1.573 2.708 1.202 3.898

sapply(data[,-2], sum)

##    a    c
## 8.40 3.22
```

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

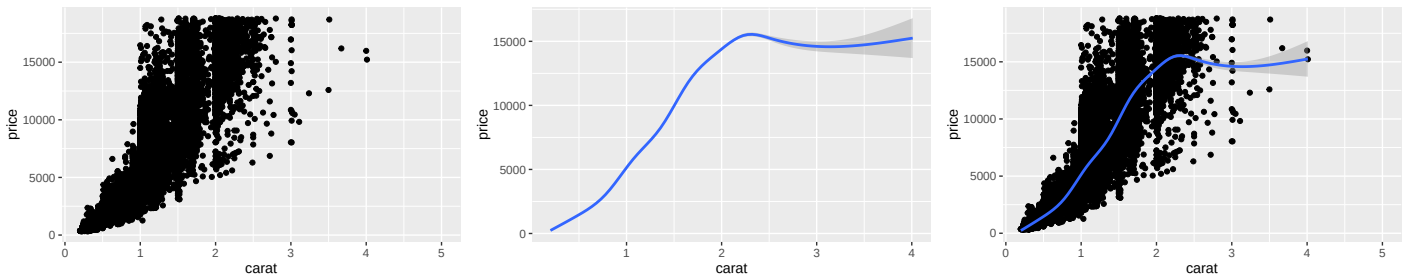
```
data[4:7, list(n, carat, price)]

##    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
## 2: 4  0.29   334       1152
## 3: 5  0.31   335       1081
## 4: 8  0.26   337       1296
## 5: 9  0.22   337       1532
```

ggplot2: An Advanced Plot for Data Frame

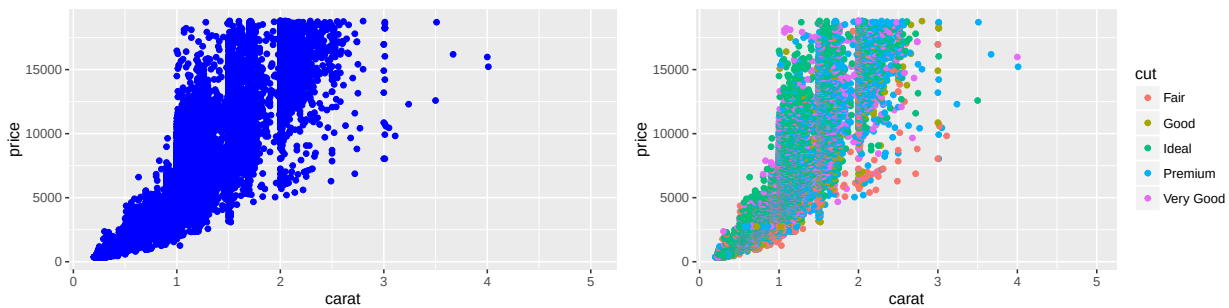
ggplot(data, mapping = aes(x = x, y = y, ...)) + geom...()

- 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 0.968 0.446

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

##          cut count  mean    sd
## 1:      Fair  1114 1.081 0.477
## 2:     Ideal 10814 0.876 0.427
## 3: Very Good  6507 0.978 0.412
## 4:      Good  2795 0.997 0.415
## 5:   Premium  7794 1.061 0.478
```

- 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  3993   326   949  2397  5302 18818

data[,lapply(.SD, summary_num)]

##        n carat cut color clarity depth table price     x     y     z
## 1: 26971 0.798  NA    NA      NA 61.75 57.46  3928  5.73  5.73  3.539
## 2: 15571 0.474  NA    NA      NA  1.43  2.23  3993  1.12  1.14  0.706
## 3:     1 0.200  NA    NA      NA 43.00 43.00   326  0.00  0.00  0.000
## 4: 13486 0.400  NA    NA      NA 61.00 56.00   949  4.71  4.72  2.910
## 5: 26971 0.700  NA    NA      NA 61.80 57.00  2397  5.70  5.71  3.530
## 6: 40455 1.040  NA    NA      NA 62.50 59.00  5302  6.54  6.54  4.040
## 7: 53940 5.010  NA    NA      NA 79.00 95.00 18818 10.74 58.90 31.800
```

Contents

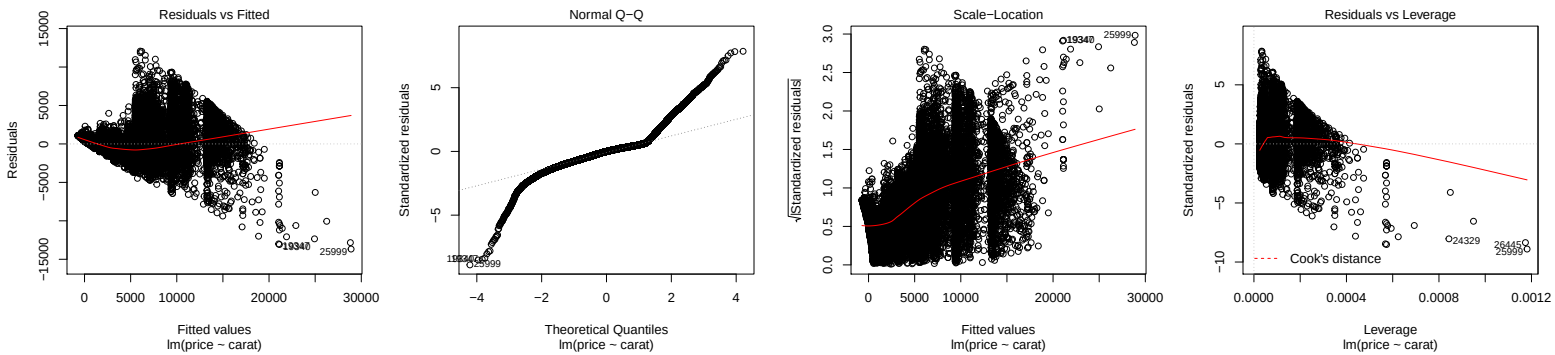
Linear Model and Residuals

- 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)
```



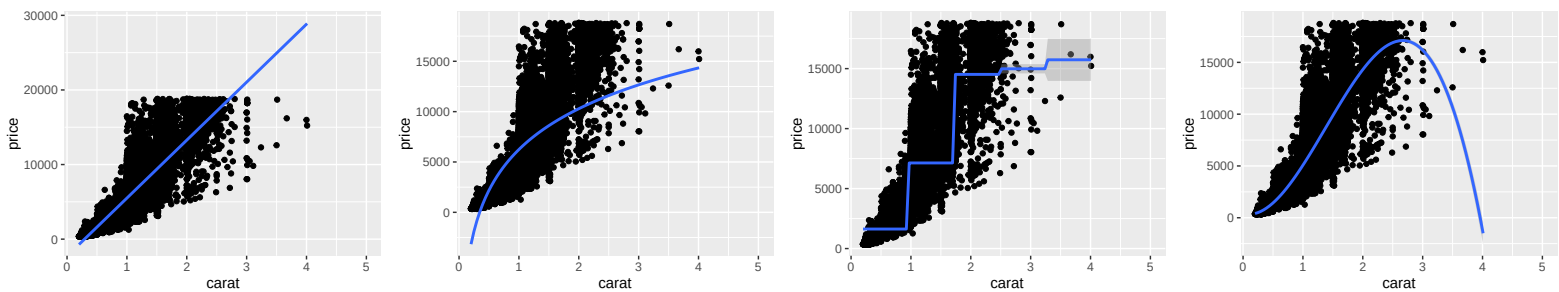
Variable Transformation

- 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 for F-test of the effect of multi variables

```
anova(model1)

## Analysis of Variance Table
##
## Response: price
##              Df   Sum Sq  Mean Sq F value Pr(>F)
## carat         1 5.39e+11 5.39e+11  243070 <2e-16 ***
## cut           4 4.39e+09 1.10e+09     495 <2e-16 ***
## carat:cut     4 1.28e+09 3.19e+08     144 <2e-16 ***
## Residuals 39698 8.81e+10 2.22e+06
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

add1(model1, ~ . + clarity, test = "F")

## Single term additions
##
## Model:
## price ~ carat + cut + carat:cut
##         Df Sum of Sq      RSS    AIC F value Pr(>F)
## <none>               8.81e+10 580237
## clarity  7  2.51e+10 6.29e+10 566901    2266 <2e-16 ***
## ---
```

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 5.39e+11 5.39e+11 4.32e+05 < 2e-16 ***
## cut           4 4.39e+09 1.10e+09 8.80e+02 < 2e-16 ***
## color         6 9.21e+09 1.54e+09 1.23e+03 < 2e-16 ***
## clarity       7 2.79e+10 3.98e+09 3.19e+03 < 2e-16 ***
## depth         1 1.66e+06 1.66e+06 1.33e+00    0.25
## table         1 6.62e+07 6.62e+07 5.30e+01 3.3e-13 ***
## x             1 2.68e+09 2.68e+09 2.15e+03 < 2e-16 ***
## Residuals 39686 4.95e+10 1.25e+06
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```