An R Lecture from Practice: Part II

Fangda Fan

2016.5

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

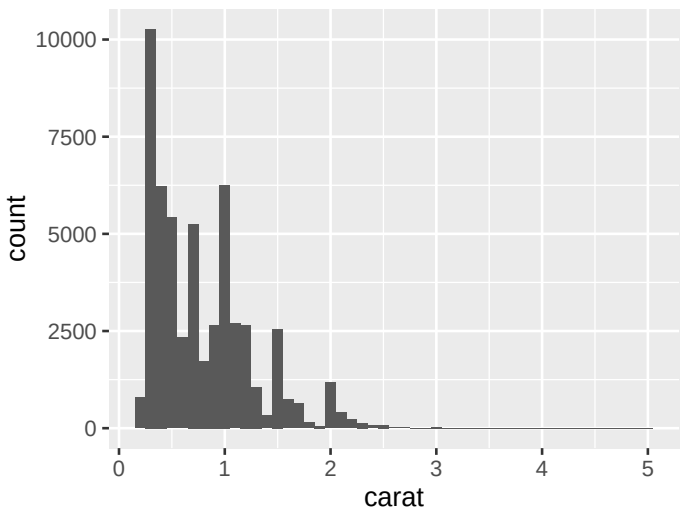
ggplot2

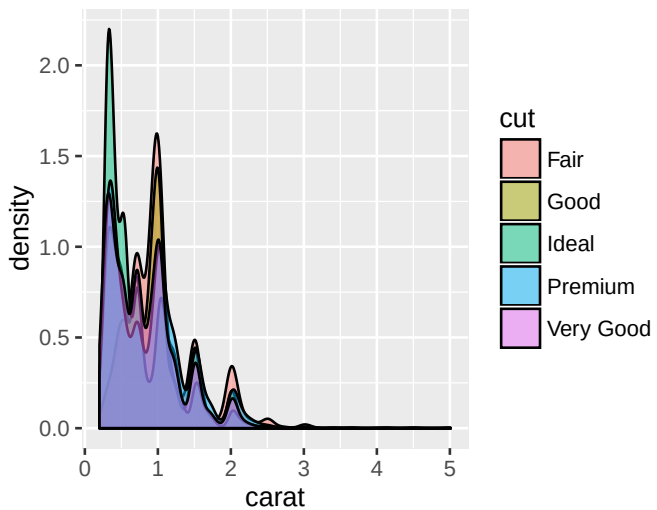
- Read data

```
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)
```



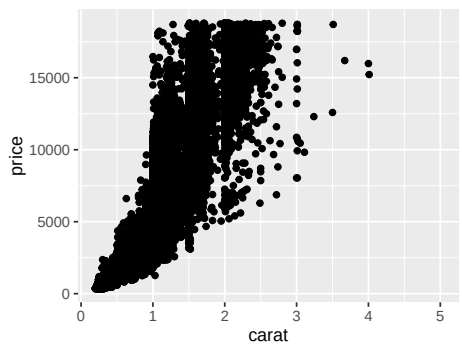


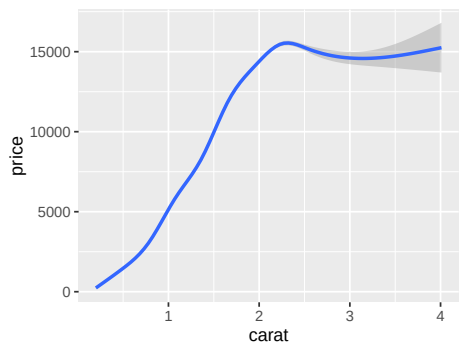
Introduction to ggplot2

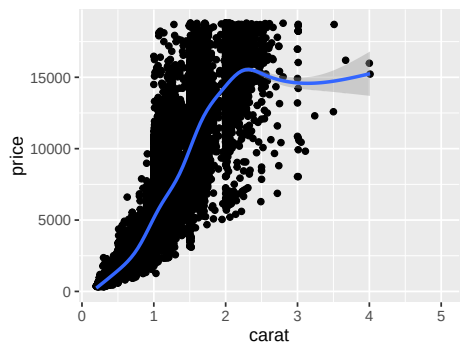
ggplot(data, mapping = aes(x = x, y = y, ...)) + geom...()

- 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()
```



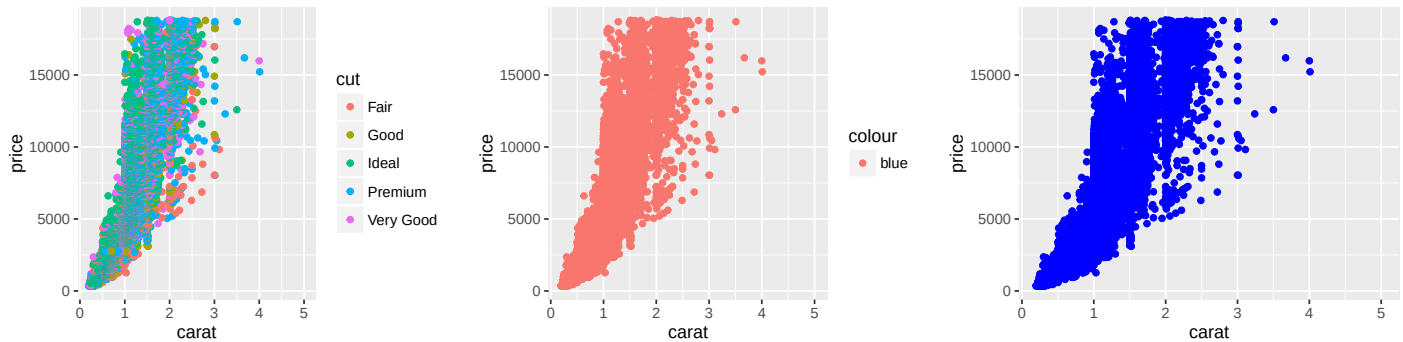




Parameters of aes()

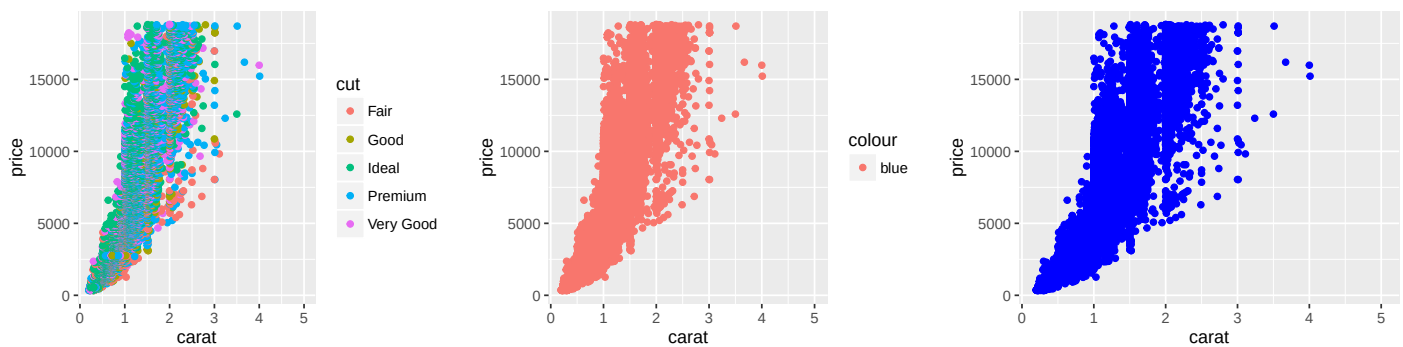
- Parameters in aes(...): use variables in data

```
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")
```

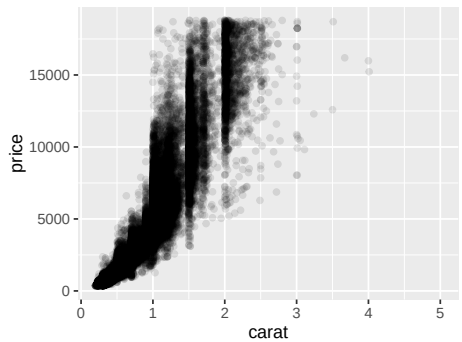


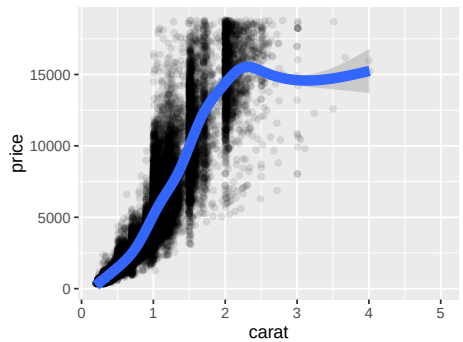
ver the sea.(Isaiah 11:9)

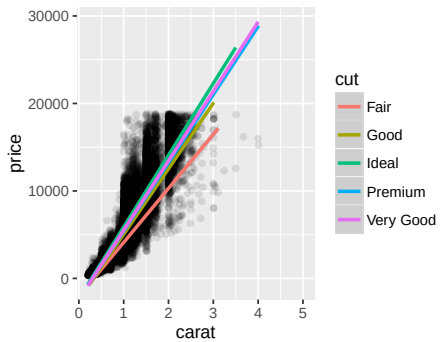
Save Plot Elements as R Variables

- 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")
```



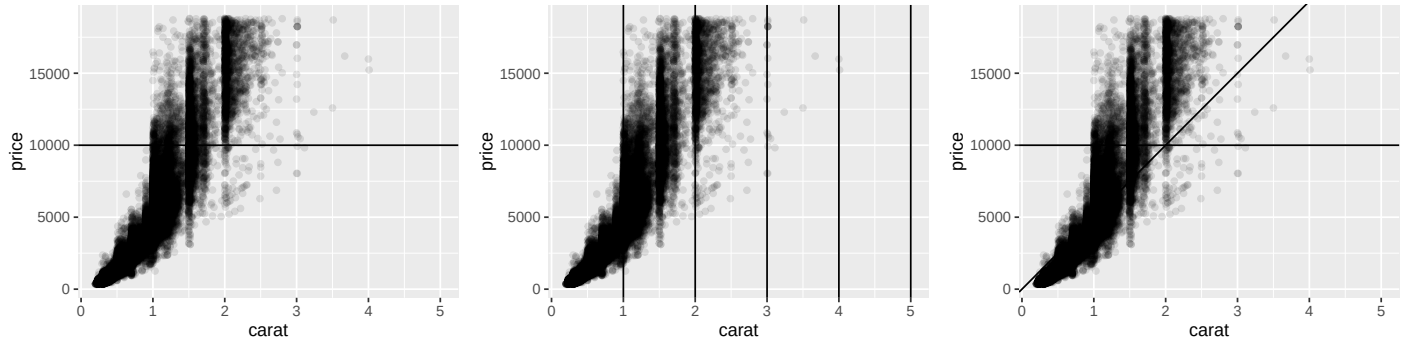




Add Lines

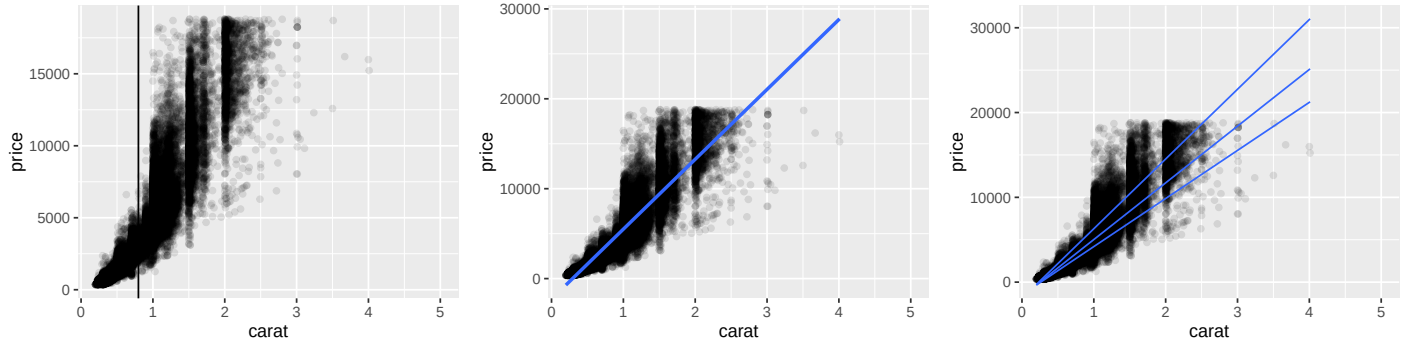
- Add straight line with geom_abline() and vertical line with geom_vline()

```
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))
```



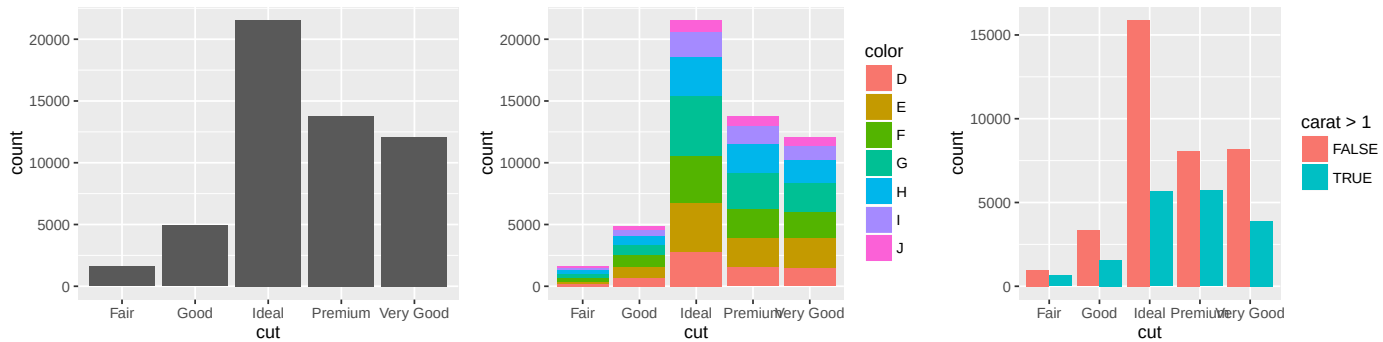
ver the sea.(Isaiah 11:9)

Contents

Univariate Plot (1): Discrete variable

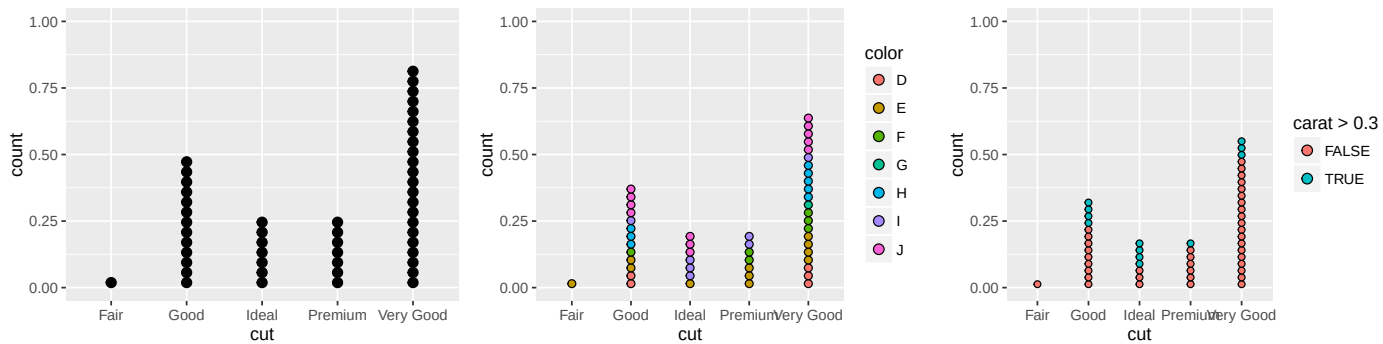
- Bar plot with geom_bar()

```
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

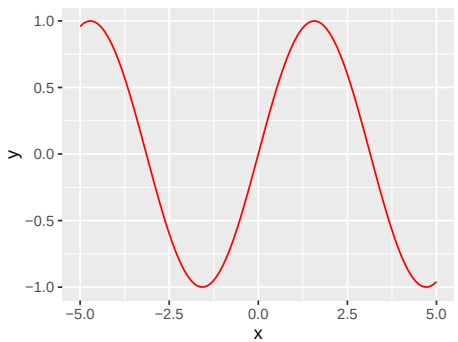
```
ggplot(data, aes(x = depth)) + stat_ecdf()
ggplot(data, aes(x = depth)) + stat_ecdf(aes(color = cut))
ggplot(data, aes(sample = depth)) + stat_qq()
ggplot(data, aes(sample = depth)) + stat_qq(aes(color = cut))
```

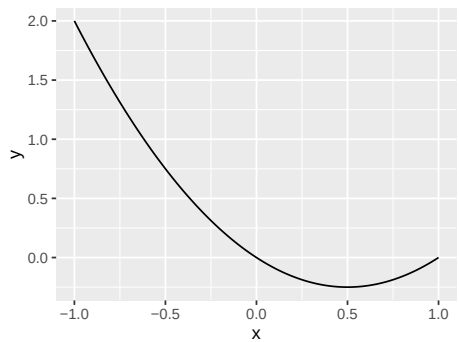
Isaiah 11:9)

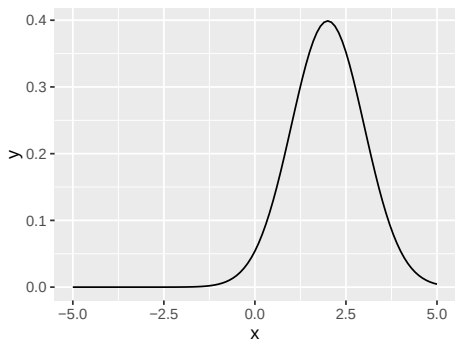
Univariate Plot (3): Function

- 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))
```



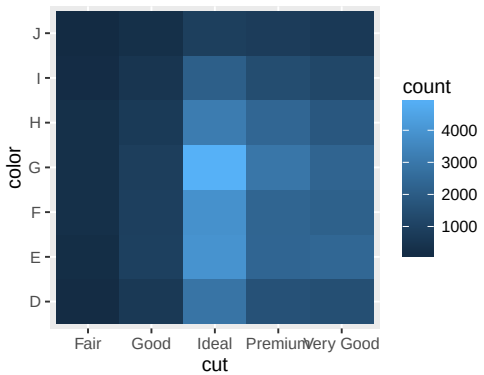


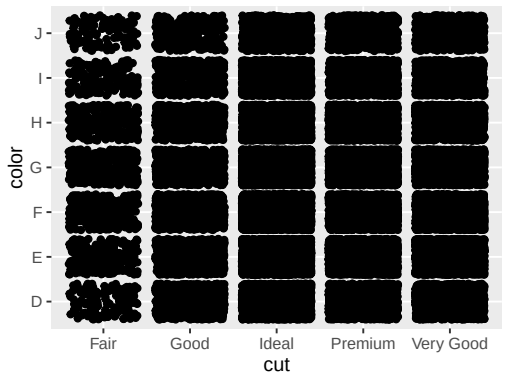


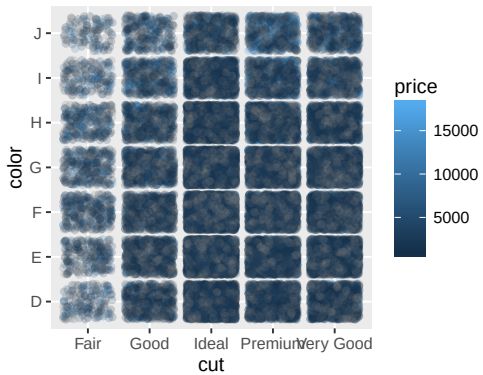
Bivariate Plot (1): Two Discrete variables

- 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)
```







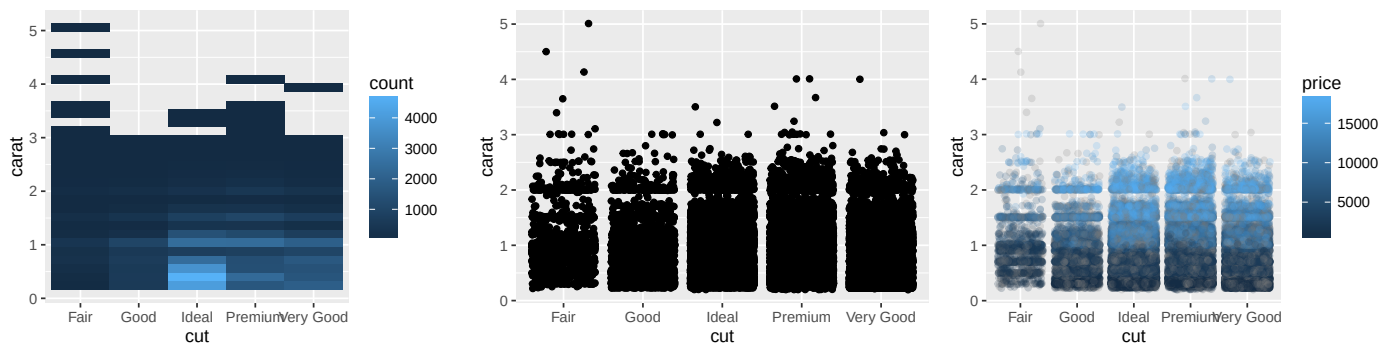
Bivariate Plot (2): A Continuous and A Discrete

- 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)
```

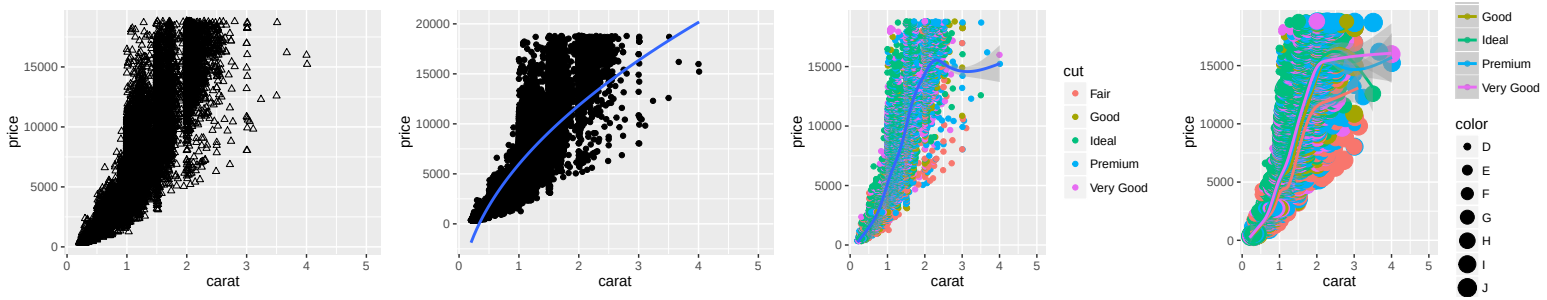


ver the sea.(Isaiah 11:9)

Bivariate Plot (3): Two Continuous Variables

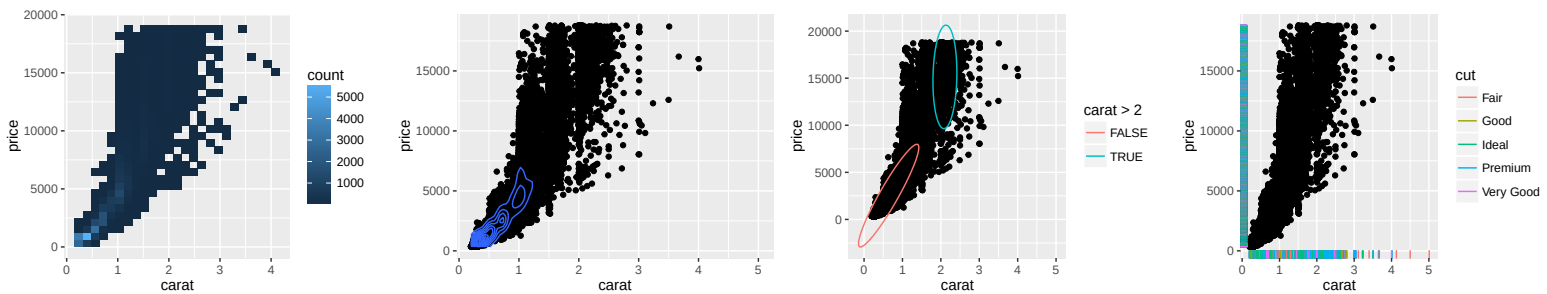
- Point plot and with smooth line

```
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))
```

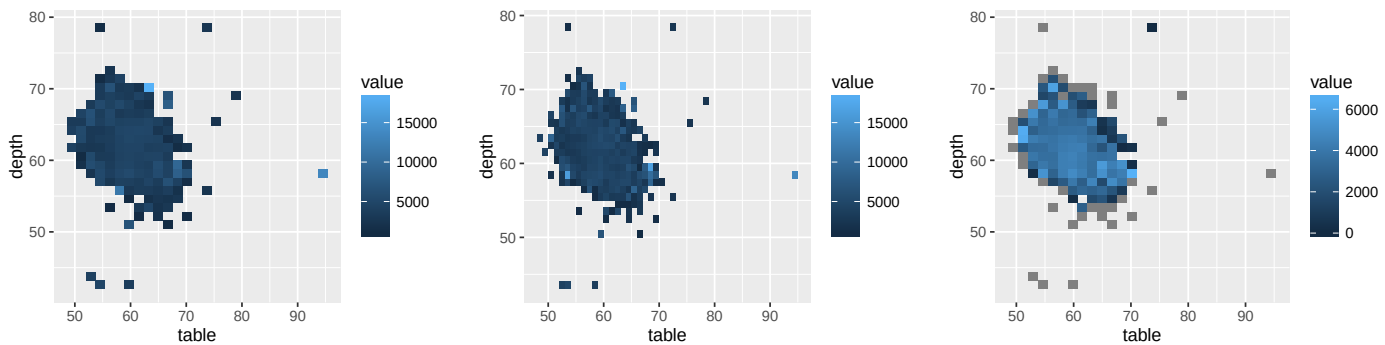


Isaiah 11:9)

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)
```

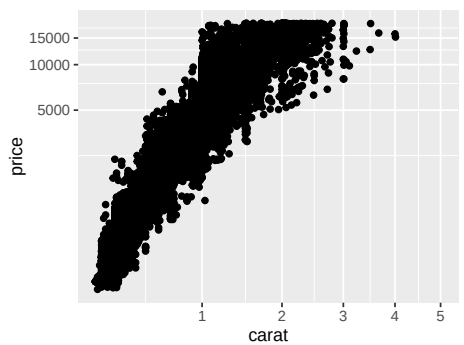


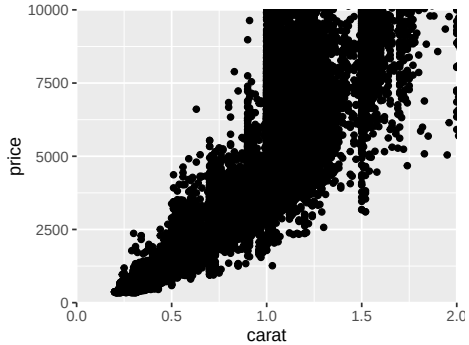
Contents

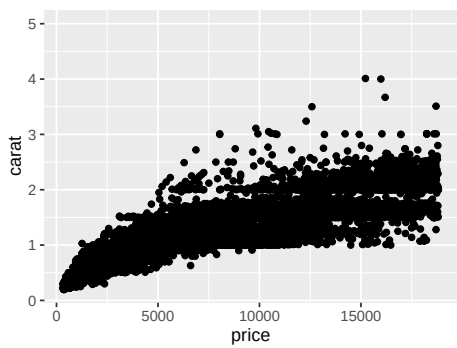
Axes, labels and themes

- 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()
```

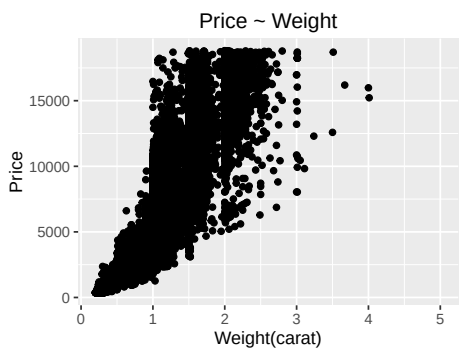


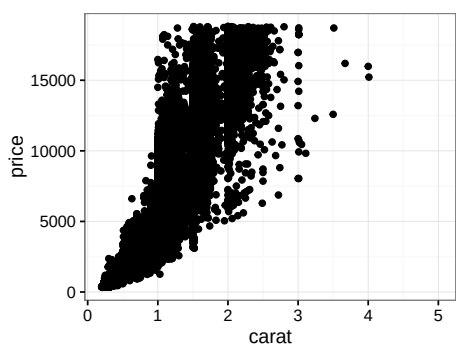


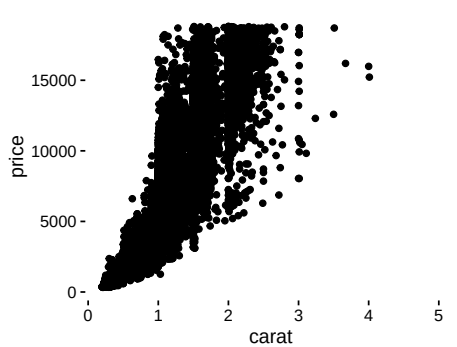


- Change title and axis labels, themes

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





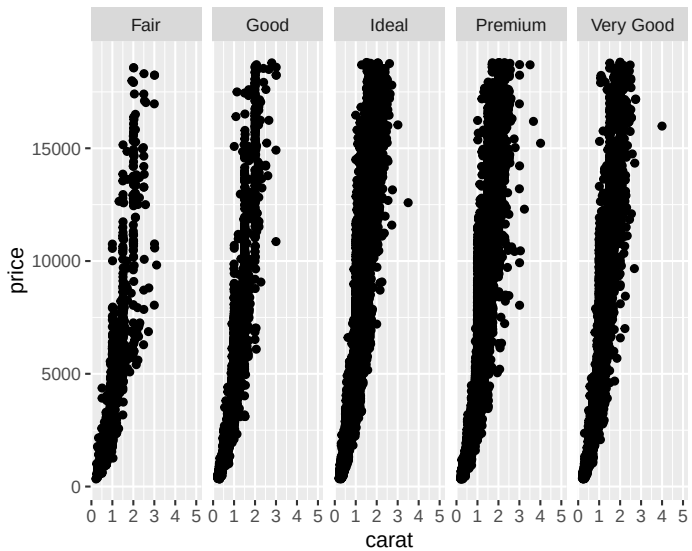


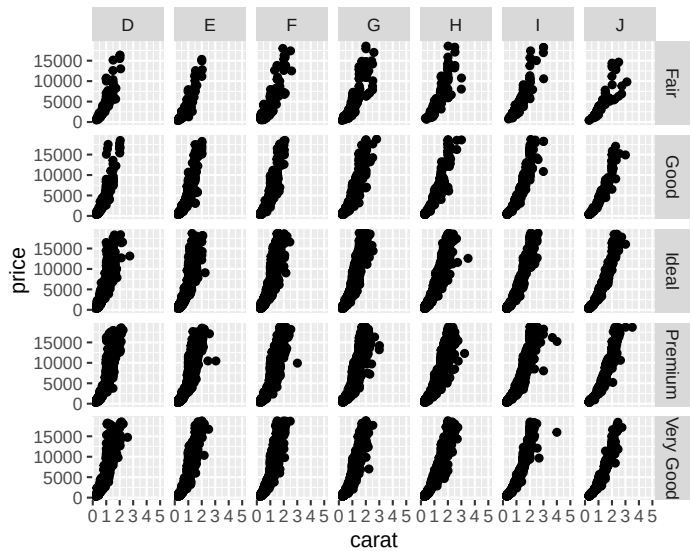
ver the sea.(Isaiah 11:9)

Facet

- Make ggplot facet with facet_grid(facets = formula)

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

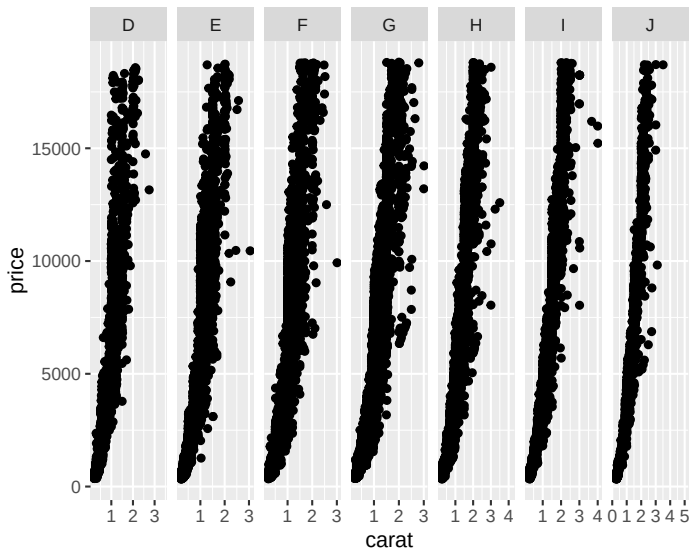




Scales and Margins in Facet

- 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.   :0.20   Length:53940       Length:53940       
##  1st Qu.:13486   1st Qu.:0.40   Class :character   Class :character   
##  Median :26970   Median :0.70   Mode  :character   Mode  :character   
##  Mean   :26970   Mean   :0.80                                         
##  3rd Qu.:40455   3rd Qu.:1.04                                         
##  Max.   :53940   Max.   :5.01                                         
##                                                                       
##    clarity              depth           table           price       
##  Length:53940       Min.   :43.0   Min.   :43.0   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       1081
##     2:      6         NA
##     3:      7         NA
##     4:      8       1296
##     5:      9       1532
##    ---
## 53932: 53936       3829
## 53933: 53937       3829
## 53934: 53938         NA
## 53935: 53939       3206
## 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 1.081
## 2:     Ideal 10814 0.876
## 3: Very Good  6507 0.978
## 4:      Good  2795 0.997
## 5:   Premium  7794 1.061
```

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
##    0.43    0.00    0.42
```

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

```
##    user  system elapsed
##    0.29    0.00    0.29
```

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  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
```

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 colorG colorH colorI
## 1:           1 1.85e-05 0.0483 0.4  0.818  0.823  0.791  0.846  0.899
##    colorJ clarityIF claritySI1 claritySI2 clarityVS1 clarityVS2
## 1:  0.948     0.967      0.758       0.83      0.849      0.773
##    clarityVVS1 clarityVVS2  depth table price       x      y      z
## 1:       0.932       0.906 0.0415 0.183 0.266 0.00831 0.0081 0.0142
```

- Delete all variables with sparsity nearly 1

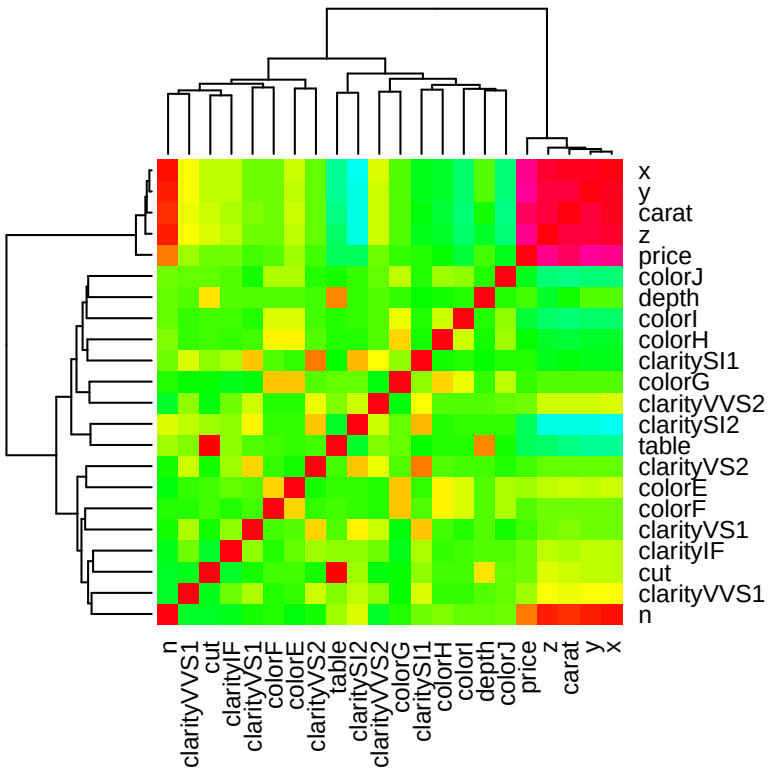
```
data_clean = data_clean[, data_clean[ ,sparse < 0.9999], with = FALSE]
dim(data_clean)

## [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))})]
```