Introduction to Data Analysis with R-project for Socioeconomics

Jittima Singvejsakul

Department of Agricultural Economy and Development, Faculty of Agriculture, Chiang Mai University

10th International Conference on Asian Economic Development

What is R program ?

- R is a wonderful tool for statistical analysis, visualization and reporting. Its usefulness is best seen in the wide variety of fields where it is used.
- Furthermore, R used by statisticians with advanced machine learning training and by programmers familiar with other languages and also by people who are not necessarily trained in advanced data analysis but are tired of using Excel.

Ross Ihaka and Robert Gentleman (1991)

Ross Ihaka and **R**obert Gentleman (1991)

```
knitr::include_graphics("Images/Ross and Robert.jpg")
```

Getting R

- Downloading R http://cran.r-project.org/.
- R Version
- Downloading R studio
  https://www.rstudio.com/products/rstudio/download/
- Choose Download free version for the R studio Desktop

1. Basic Math

R can certainly be used to do basic math.

```
1 + 1
```

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

```
3 * 7 * 2
```

```
## [1] 42
```

2.Variable Assignment

There are a number of ways to assign a value to a variable and the valid assignment operators are assign.

```
x <- 2
x
```

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

```
assign("j", 4)
j
```

```
## [1] 4
```

3.Vectors

A vector is a collection of elements, all of the same type. For instance, c(1,3, 2, 1, 5) is a vector consisting of the numbers 1, 3, 2, 1, 5, in that order.

```
x <- c(1,4,6,9,3,2)
x
```

```
## [1] 1 4 6 9 3 2
```

1.data.frames

On the surface a data.frame is just like an Excel spreadsheet in that it has columns and rows. In statistical terms, each column is a variable and each row is an observation.

```
x <- 10:1
y <- -4:5
q <- c("Hockey", "Football", "Baseball", "Curling",
       "Rugby","Lacrosse", "Basketball", "Tennis",
       "Cricket","Soccer")
```

```
theDF <- data.frame(x, y, q)
theDF
```

```
##     x  y          q
## 1  10 -4     Hockey
## 2   9 -3   Football
## 3   8 -2   Baseball
## 4   7 -1    Curling
## 5   6  0      Rugby
## 6   5  1   Lacrosse
## 7   4  2 Basketball
## 8   3  3     Tennis
## 9   2  4    Cricket
## 10  1  5     Soccer
```

```
theDF <- data.frame(First=x, Second=y, Sport=q)
theDF
```

```
##    First Second      Sport
## 1     10     -4     Hockey
## 2      9     -3   Football
## 3      8     -2   Baseball
## 4      7     -1    Curling
## 5      6      0      Rugby
## 6      5      1   Lacrosse
## 7      4      2 Basketball
## 8      3      3     Tennis
## 9      2      4    Cricket
## 10     1      5     Soccer
```

2.Matrices

A very common mathematical structure that is essential to statistics is a matrix. This is similar to a data.frame in that it is rectangular with rows and columns except that every single element, regardless of column, must be the same type, most commonly all numerics.

```
A <- matrix(1:10, nrow=5)
A
```

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

```r
B <- matrix(21:30, nrow=5, byrow= TRUE)
B
```

```
##      [,1] [,2]
## [1,]   21   22
## [2,]   23   24
## [3,]   25   26
## [4,]   27   28
## [5,]   29   30
```

cbind

The cbind function - short for column bind - is a merge function that can be used to combine two data frames with the same number of multiple rows into a single data frame.

```
cbind(A,B)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    6   21   22
## [2,]    2    7   23   24
## [3,]    3    8   25   26
## [4,]    4    9   27   28
## [5,]    5   10   29   30
```

rbind

The name of the rbind R function stands for row-bind. The rbind function can be used to combine several vectors, matrices and/or data frames by rows.

```
rbind(A,B)
```

```
##        [,1] [,2]
##  [1,]    1    6
##  [2,]    2    7
##  [3,]    3    8
##  [4,]    4    9
##  [5,]    5   10
##  [6,]   21   22
##  [7,]   23   24
##  [8,]   25   26
##  [9,]   27   28
## [10,]   29   30
```

3.Arrays

An array is essentially a multidimensional vector. It must all be of the same type, and individual elements are accessed in a similar fashion using square brackets. The first element is the row index,the second is the column index and the remaining elements are for outer dimensions.

```
theArray <- array(1:12, dim=c(2, 3, 2))
theArray
```

```
## , , 1
##
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
##
## , , 2
##
##      [,1] [,2] [,3]
## [1,]    7    9   11
## [2,]    8   10   12
```

Setting The Working Directory In R

getwd

getwd() returns an absolute filepath representing the current working directory. This result will be a character string. It can return Null if the working directory is not available.

setwd

To change the current working directory, use the setwd R function.The function requires the new working directory as an argument to the function. You can define this in absolute terms (a specific path).

Setting The Working Directory In R {code}

- getwd( )
- setwd("/my/new/ path)
- dir( )

Reading Data into R

- CSVs

```
theUrl <-"http://www.jaredlander.com/data/TomatoFirst.csv"
tomato <-read.table(file=theUrl, header=TRUE, sep=",")
head(tomato)
```

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

Reading Data into R

- Text file

read.table (file location,header =T , sep = “”)

- Histrogram

hist(x)

```
library(ggplot2)
```

```
## Registered S3 methods overwritten by 'tibble':
##   method     from
##   format.tbl pillar
##   print.tbl  pillar
```

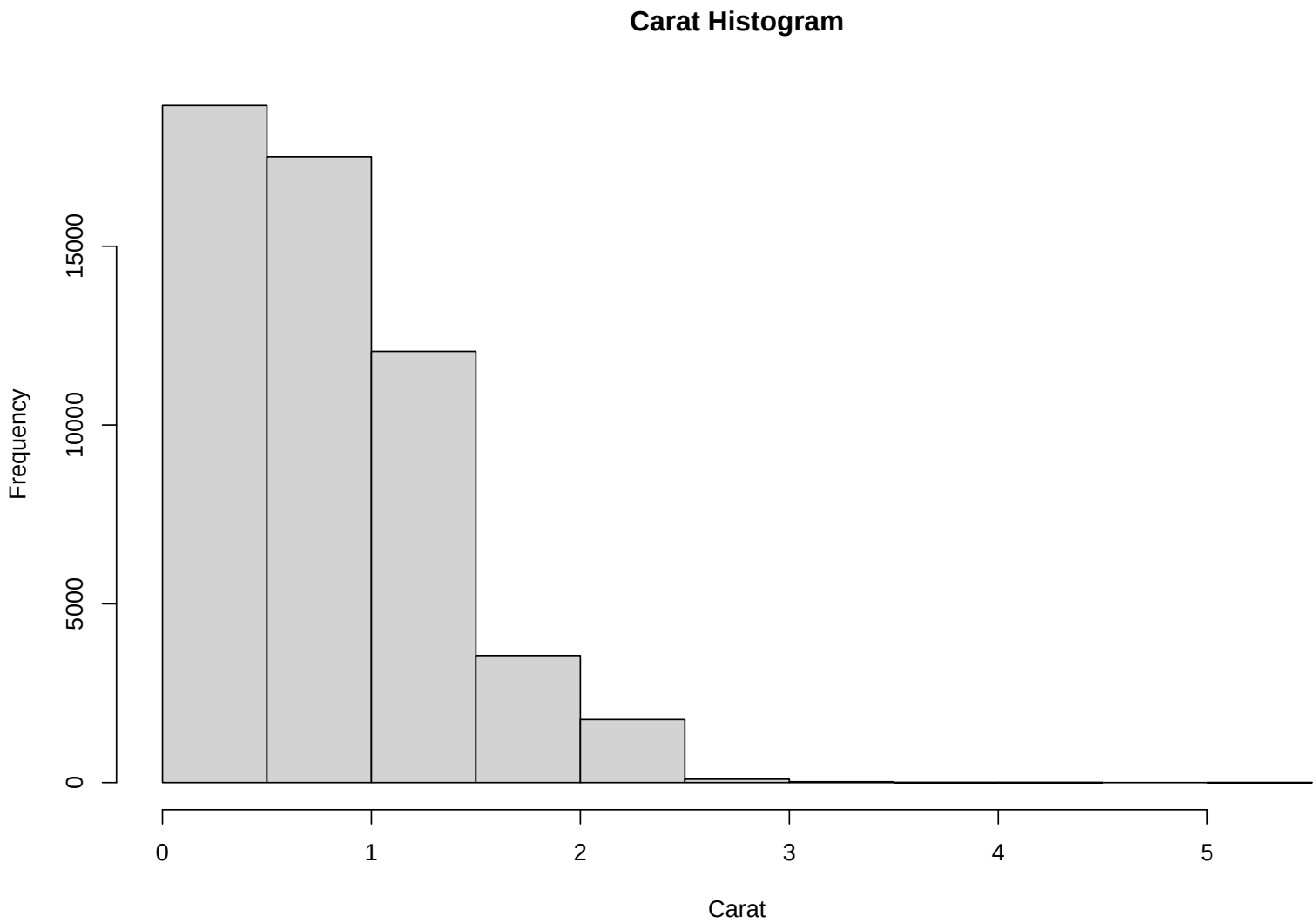
```
data(diamonds)
head(diamonds)
```

```
## Warning: `...` is not empty.
##
## We detected these problematic arguments:
## * `needs_dots`
##
## These dots only exist to allow future extensions and should be empty
## Did you misspecify an argument?
```

```
## # A tibble: 6 x 10
##   carat cut       color clarity depth table price     x     y     z
```

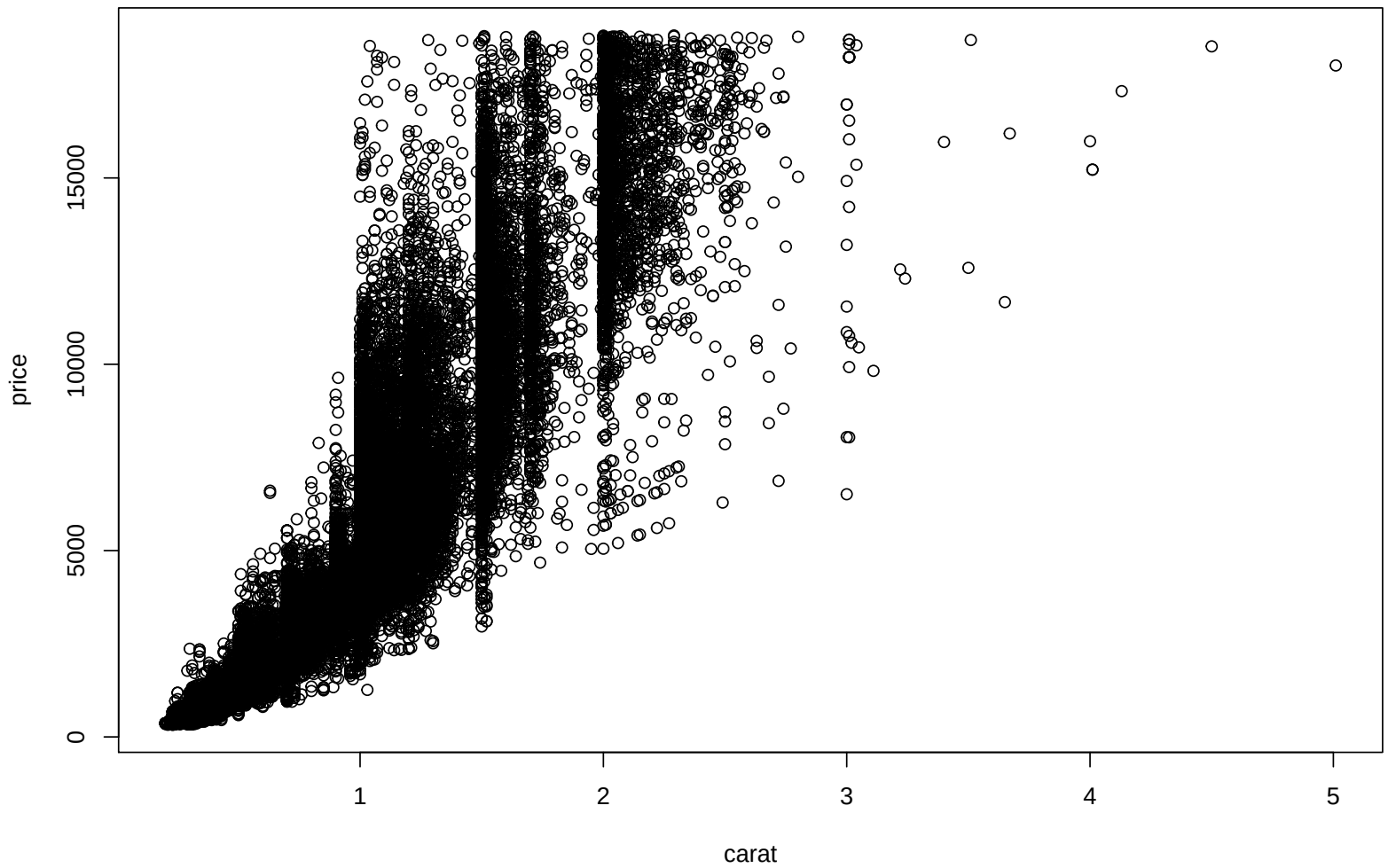
Statistical Graphics {Histrogram}

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



Statistical Graphics {Scatterplot}

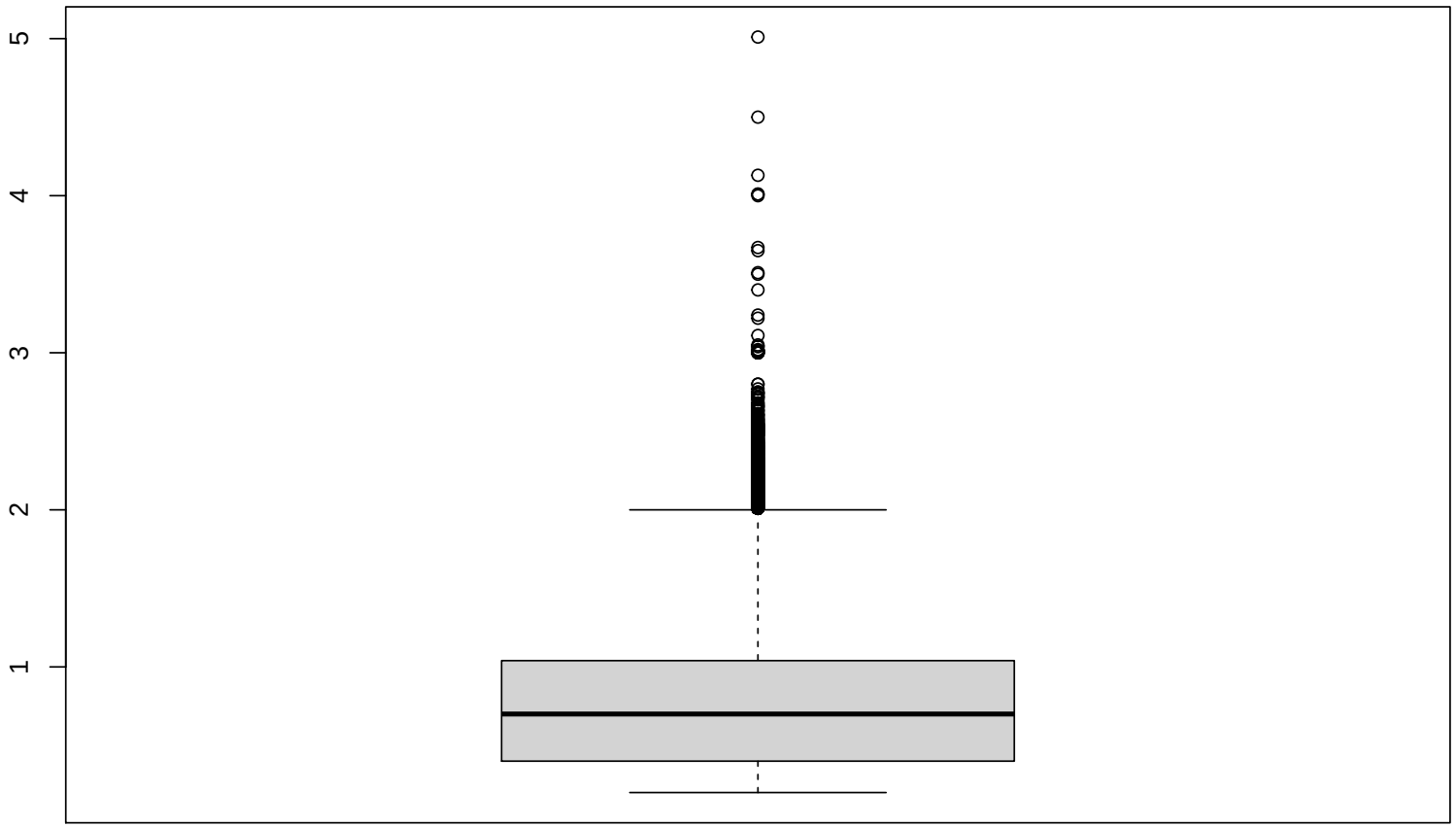
```
plot(price ~ carat, data=diamonds)
```




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

Statistical Graphics {Boxplots}

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

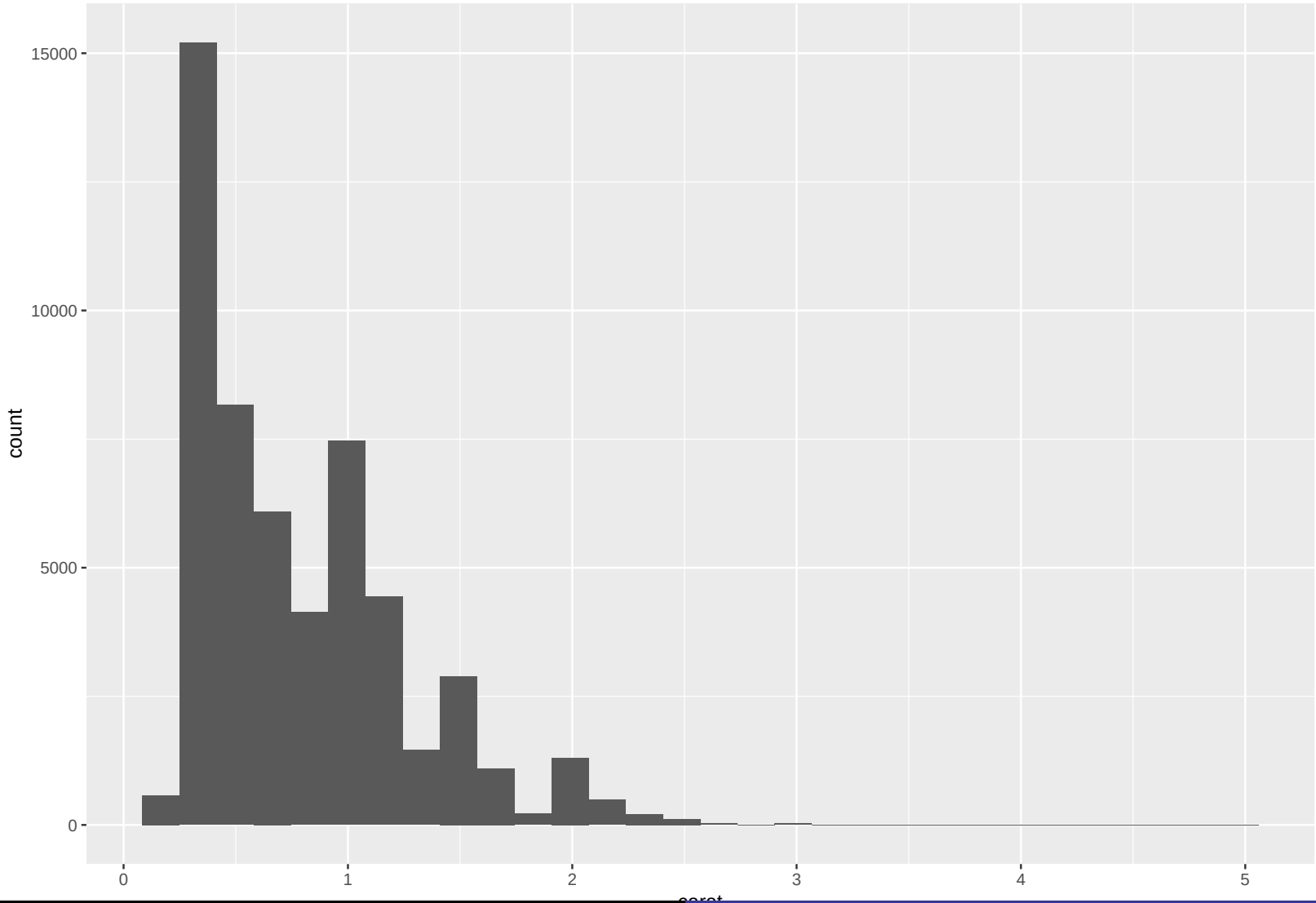


ggplot2

ggplot2 is a system for declaratively creating graphics, based on The Grammar of Graphics. You provide the data, tell ggplot2 how to map variables to aesthetics, what graphical primitives to use, and it takes care of the details.(https://ggplot2.tidyverse.org/)

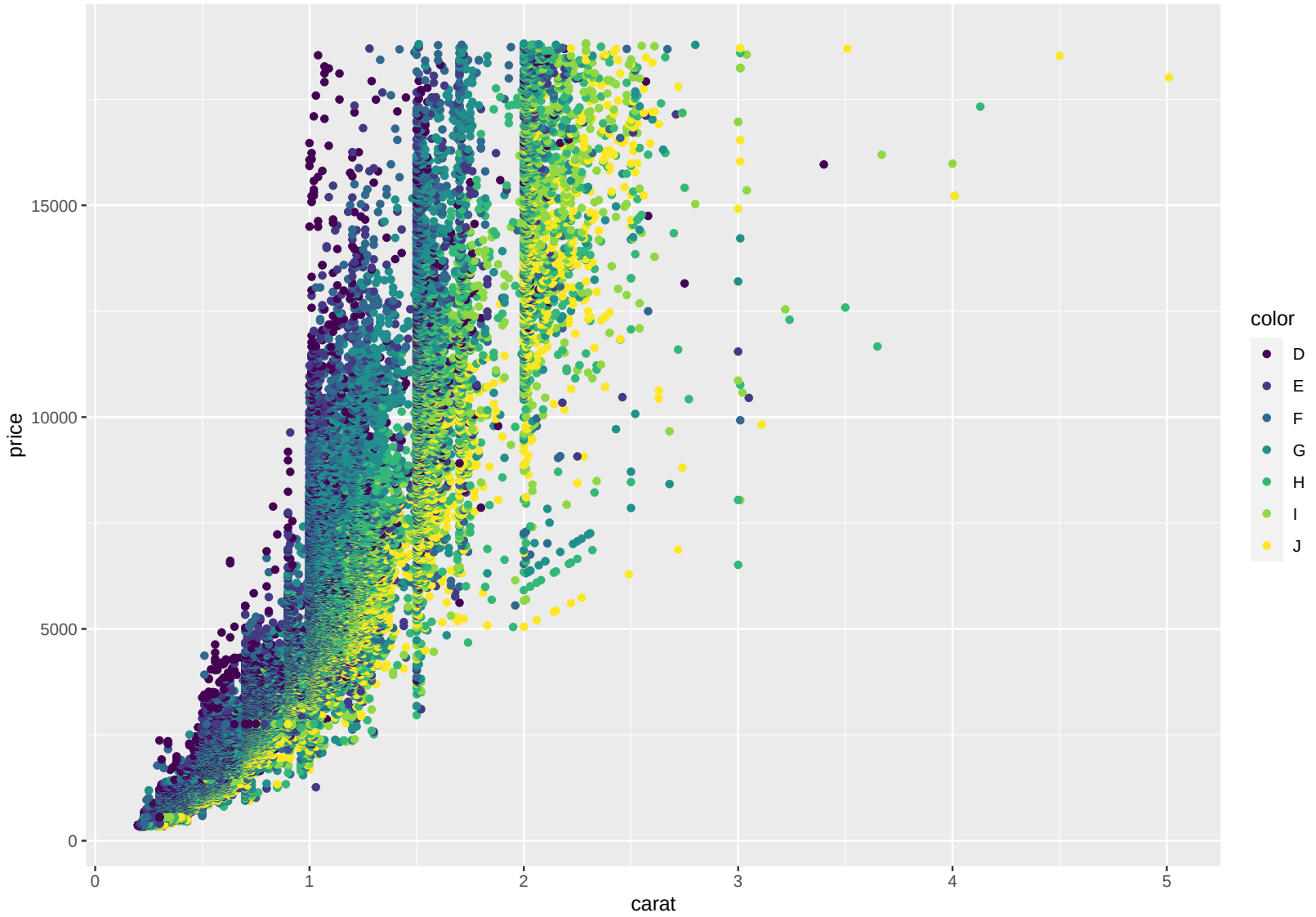
```
ggplot(data=diamonds) + geom_histogram(aes(x=carat))
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



```
g <-ggplot(diamonds,aes(x=carat, y=price))
```

```
g + geom_point(aes(color=color))
```



Writing R functions

R functions

R has a convenient way to make functions,but it is very different from other languages, so some expectation adjustmentmight be necessary.

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

```
## [1] "Hello Jared"
```

Return Values

Functions are generally used for computing some value, so they need a mechanism to supply that value back to the caller. This is called returning and is done quite easily.The return command more explicitly specifies that a value should be returned and the function should be exited.

```
double.num <- function(x)
 {
 return(x * 2)
 }
double.num(5)
```

```
## [1] 10
```

if and else

The most common test is the if command. It essentially says. If something is TRUE, then perform some action; otherwise, do not perform that action.

Control Statements

```
check.bool <- function(x)
{
if(x == 1)
{
# if the input is equal to 1, print hello
print("hello")
}else if(x == 0)
{
# if the input is equal to 0, print goodbye
print("goodbye")
}else
{
# otherwise print confused
print("confused")
}
}
```

Control Statements

```
check.bool(1)
```

```
## [1] "hello"
```

```
check.bool(0)
```

```
## [1] "goodbye"
```

```
check.bool(2)
```

```
## [1] "confused"
```

Loops, the Un-R Way to Iterate

for Loops

The most commonly used loop is the for loop. It iterates over an index-provided as a vector-and performs some operations. For a first simple example, we print out the first ten numbers. The loop is declared using for, which takes one English-seeming argument in three parts. The third part is any vector of values of any kind, most commonly numeric or character. The first part is the variable that is iteratively assigned the values in the vector from the third part.

Loops, the Un-R Way to Iterate {for Loops}

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

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

Loops, the Un-R Way to Iterate {while Loops}

```
x <- 1
while(x <= 5)
{
print(x)
x <- x + 1
}
```

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

Sometimes we have to skip to the next iteration of the loop or completely break out of it.

```
for(i in 1:10)
{
if(i == 4)
{
break
}
print(i)
}
```

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

Apply is the first member of this family that users usually learn, and it is also the most restrictive. It must be used on a matrix, meaning all of the elements must be of the same type whether they are character, numeric or logical.

```
# build the matrix
theMatrix <- matrix(1:9, nrow=3)
# sum the rows
apply(theMatrix, 1, sum)
```

```
## [1] 12 15 18
```

Lapply works by applying a function to each element of a list and returning the results as a list.

```
theList <- list(A=matrix(1:9, 3), B=1:5, C=2)
lapply(theList, sum)
```

```
## $A
## [1] 45
##
## $B
## [1] 15
##
## $C
## [1] 2
```

Dealing with lists can be cumbersome, so to return the result of lapply as a vector instead, use **sapply**. It is exactly the same as lapply in every other way.

```
sapply(theList, sum)
```

```
##  A  B  C
## 45 15  2
```

Probability distributions lie at the heart of statistics, so naturally R provides numerous functions for making use of them. These include functions for generating random numbers and calculating the distribution and quantile.

1. Normal Distribution

   - To draw random numbers from the **normal distribution** use the **rnorm** function and **dnorm** function, which optionally allows the specification of the mean and standard deviation.

```
# 10 draws from the 100-20 distribution
rnorm(n=10, mean=100, sd=20)
```

```
##  [1] 133.44491  83.34295  85.81036 126.41819  55.98335  73.72270 131
##  [8]  92.01681  94.65221 115.76072
```

2. Binomial Distribution

- Generating random numbers from the binomial distribution is not simply generating random numbers but rather generating the number of successes of independent trials. To simulate the number of successes out of ten trials with probability 0.4 of success, we run **rbinom** with n=1 (only one run of the trials), size=10 (trial size of 10) and prob=0.4 (probability of success is 0.4).

```
rbinom(n=1, size=10, prob=.4)
```

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

```
rbinom(n=10, size=10, prob=.4)
```

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

Other distributions

```
knitr::include_graphics("Images/R distributions.png")
```

Distribution	Random Number	Density	Distribution	Quantile
Normal	rnorm	dnorm	pnorm	qnorm
Binomial	rbinom	dbinom	pbinom	qbinom
Poisson	rpois	dpois	ppois	qpois
t	rt	dt	pt	qt
F	rf	df	pf	qf
Chi-Squared	rchisq	dchisq	pchisq	qchisq
Gamma	rgamma	dgamma	pgamma	qgamma
Geometric	rgeom	dgeom	pgeom	qgeom
Negative Binomial	rnbinom	dnbinom	pnbinom	qnbinom
Exponential	rexp	dexp	pexp	qexp
Weibull	rweibull	dweibull	pweibull	qweibull
Uniform (Continuous)	runif	dunif	punif	qunif
Beta	rbeta	dbeta	pbeta	qbeta
Cauchy	rcauchy	dcauchy	pcauchy	qcauchy
Multinomial	rmultinom	dmultinom	pmultinom	qmultinom
Hypergeometric	rhyper	dhyper	phyper	qhyper
Log-normal	rlnorm	dlnorm	plnorm	qlnorm
Logistic	rlogis	dlogis	plogis	qlogis

```
x <- sample(x=1:100, size=100, replace=TRUE)
x
```

```
##   [1]  48  11  73  62  36  19  24  65  14  14  21   7  25  28  46   3
##  [19]  91  40  58  76   2  54  39  19  92  55  49  33   9  65  52   7
##  [37]  39  44  56  21  58  22  65   6  74  28  75   3  83  57   2   8
##  [55]  79  41  87  52  72  67  29  54  89  76  23  26  48  66   6  10
##  [73]  93  70   5  84  97  35  81  64  56   5  71  71  73  78  18   5
##  [91]  98  25  42  38  15  29  32  15  61  16
```

```
mean(x)
```

```
## [1] 46.85
```

```
var(x)
```

```
## [1] 764.6742
```

```
sqrt(var(x))
```

```
## [1] 27.65274
```

```
sd(x)
```

```
## [1] 27.65274
```

```
min(x)
```

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

```
max(x)
```

```
## [1] 100
```

```
median(x)
```

```
## [1] 48
```

```
summary(x)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    2.00   22.75   48.00   46.85   70.25  100.00
```

```
# calculate the 25th and 75th quantile
quantile(x, probs=c(.25, .75))
```

```
##   25%   75%
## 22.75 70.25
```

```
library(ggplot2)
head(economics)
```

```
## Warning: `...` is not empty.
##
## We detected these problematic arguments:
## * `needs_dots`
##
## These dots only exist to allow future extensions and should be empty
## Did you misspecify an argument?
```

```
## # A tibble: 6 x 6
##   date         pce    pop psavert uempmed unemploy
##   <date>     <dbl>  <dbl>   <dbl>   <dbl>    <dbl>
## 1 1967-07-01  507. 198712    12.6     4.5     2944
## 2 1967-08-01  510. 198911    12.6     4.7     2945
## 3 1967-09-01  516. 199113    11.9     4.6     2958
## 4 1967-10-01  512. 199311    12.9     4.9     3143
## 5 1967-11-01  517. 199498    12.8     4.7     3066
## 6 1967-12-01  525. 199657    11.8     4.8     3018
```

```r
# use cor to calculate correlation
cor(economics$pce, economics$psavert)
```

```
## [1] -0.7928546
```

```r
# calculate each part of correlation
xPart <- economics$pce - mean(economics$pce)
yPart <- economics$psavert - mean(economics$psavert)
nMinusOne <- (nrow(economics) - 1)
xSD <- sd(economics$pce)
ySD <- sd(economics$psavert)
# use correlation formula
sum(xPart * yPart) / (nMinusOne * xSD * ySD)
```

```
## [1] -0.7928546
```

Simple Linear Regression

In its simplest form regression is used to determine the relationship between two variables. That is, given one variable, it tells us what we can expect from the other variable. This powerful tool, which is frequently taught and can accomplish a great deal of analysis with minimal effort, is called simple linear regression.

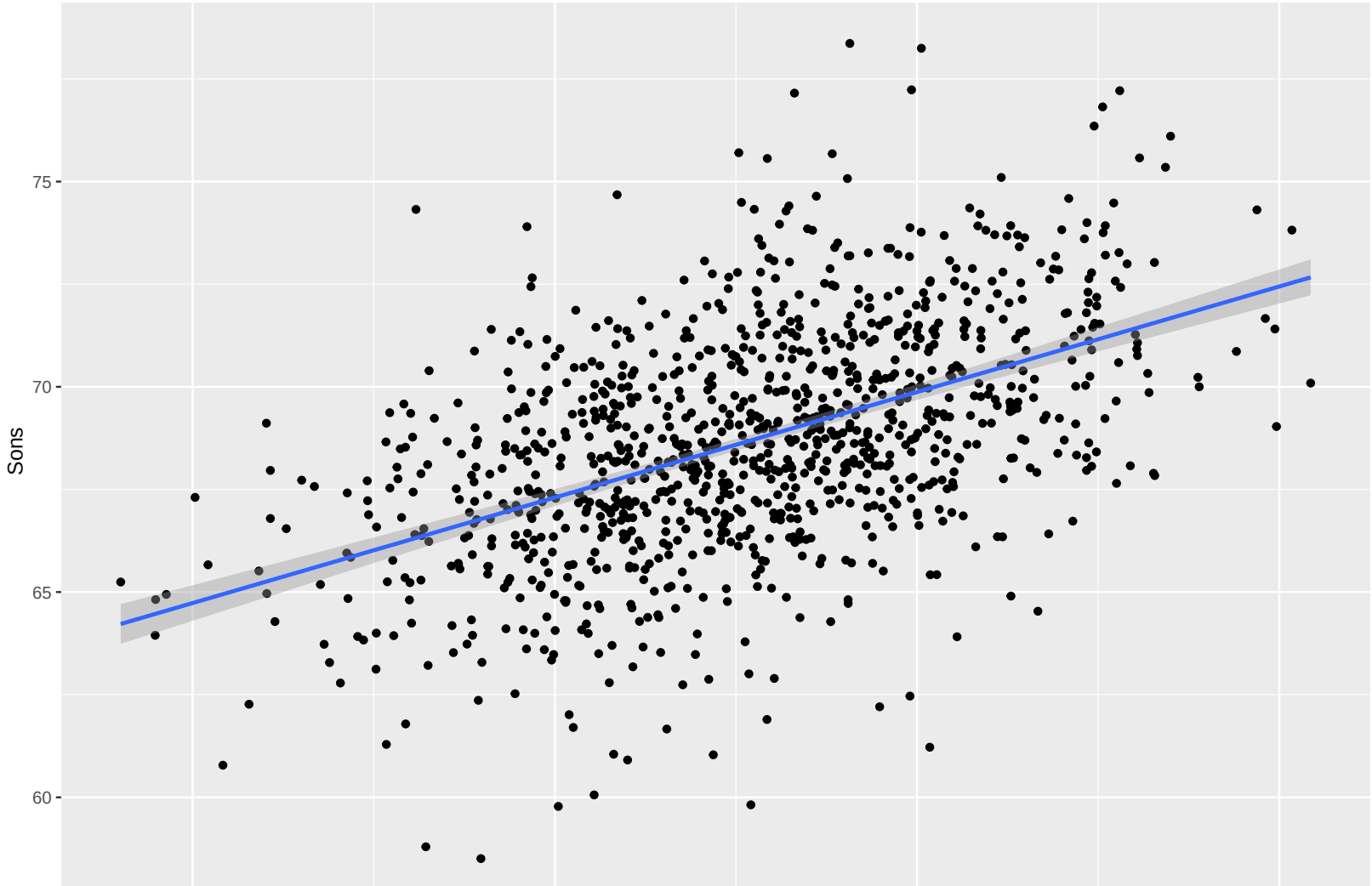
For example, fathers' heights are used as the predictor and the sons' heights as the response. The blue line running through the points is the regression line and the gray band around i t represents the uncertainty in the fit.

```r
data(father.son, package='UsingR')
head(father.son)
```

```
##    fheight  sheight
## 1 65.04851 59.77827
## 2 63.25094 63.21404
## 3 64.95532 63.34242
## 4 65.75250 62.79238
## 5 61.13723 64.28113
## 6 63.02254 64.24221
```

```
ggplot(father.son, aes(x=fheight, y=sheight)) + geom_point() +
 geom_smooth(method="lm") + labs(x="Fathers",y="Sons")
```

```
## `geom_smooth()` using formula 'y ~ x'
```



```
heightsLM <- lm(sheight ~ fheight, data=father.son)
heightsLM
```

```
##
## Call:
## lm(formula = sheight ~ fheight, data = father.son)
##
## Coefficients:
## (Intercept)      fheight
##     33.8866       0.5141
```

```
summary(heightsLM)
```

```
##
## Call:
## lm(formula = sheight ~ fheight, data = father.son)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -8.8772 -1.5144 -0.0079  1.6285  8.9685
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 33.88660    1.83235   18.49   <2e-16 ***
## fheight      0.51409    0.02705   19.01   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.437 on 1076 degrees of freedom
## Multiple R-squared:  0.2513, Adjusted R-squared:  0.2506
## F-statistic: 361.2 on 1 and 1076 DF,  p-value: < 2.2e-16
```

```
summary(heightsLM)
```

```
##
## Call:
## lm(formula = sheight ~ fheight, data = father.son)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -8.8772 -1.5144 -0.0079  1.6285  8.9685
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 33.88660    1.83235   18.49   <2e-16 ***
## fheight      0.51409    0.02705   19.01   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.437 on 1076 degrees of freedom
## Multiple R-squared:  0.2513, Adjusted R-squared:  0.2506
## F-statistic: 361.2 on 1 and 1076 DF,  p-value: < 2.2e-16
```

- Multiple Regression

```
housing <-read.table("http://www.jaredlander.com/data/housing.csv",
          sep = ",", header = TRUE,stringsAsFactors = FALSE)
names(housing) <- c("Neighborhood", "Class",
        "Units","YearBuilt","SqFt", "Income",
        "IncomePerSqFt","Expense","ExpensePerSqFt",
 "NetIncome","Value","ValuePerSqFt", "Boro")
house1 <- lm(ValuePerSqFt ~ Units + SqFt + Boro,
             data=housing)
```

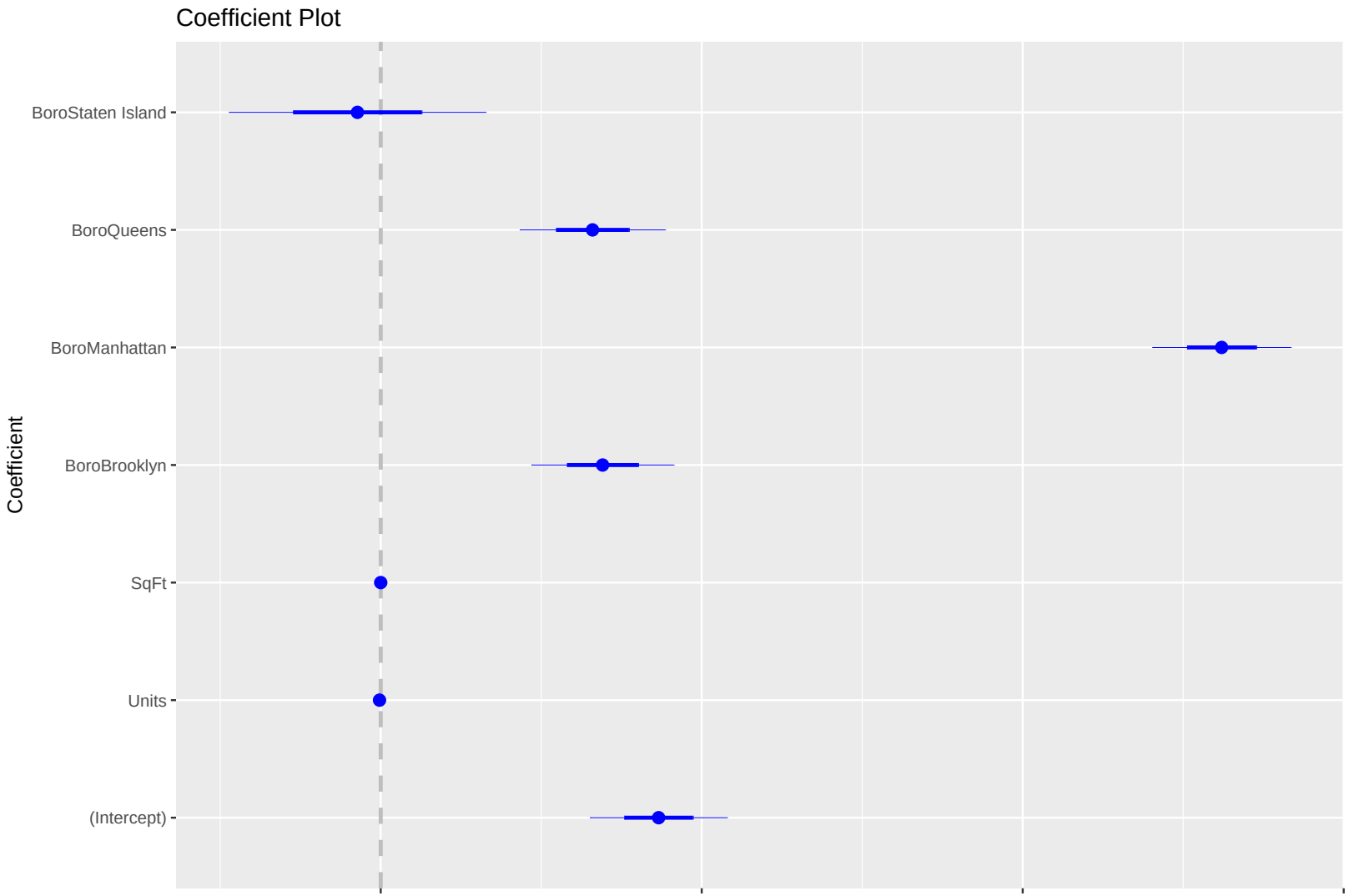
- Multiple Regression

```
summary(house1)
```

```
##
## Call:
## lm(formula = ValuePerSqFt ~ Units + SqFt + Boro, data = housing)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -164.418  -22.692    1.416   26.972  261.122
##
## Coefficients:
##                    Estimate Std. Error t value Pr(>|t|)
## (Intercept)       4.329e+01  5.330e+00   8.122 6.97e-16 ***
## Units            -1.881e-01  2.210e-02  -8.511  < 2e-16 ***
## SqFt              2.103e-04  2.087e-05  10.079  < 2e-16 ***
## BoroBrooklyn      3.456e+01  5.535e+00   6.244 4.95e-10 ***
## BoroManhattan     1.310e+02  5.385e+00  24.327  < 2e-16 ***
## BoroQueens        3.299e+01  5.663e+00   5.827 6.35e-09 ***
## BoroStaten Island -3.630e+00  9.993e+00  -0.363    0.716
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

- Multiple Regression

```
library(coefplot)
coefplot(house1)
```



Logistic Regression

Not all data can be appropriately modeled with linear regression, because they are binomial (TRUE/FALSE), count data or some other form. To model these types of data, generalized linear models were developed. They are still modeled using a linear predictor, X??, but they are transformed using some link function.

The examples in this section will use the a subset of data from the 2010 American Community Survey (ACS) for New York State.1 ACS data contain a lot of information, so we have made a subset of it with 22,745 rows and 18 columns.

- Logistic Regression

```
acs <-read.table("http://jaredlander.com/data/acs_ny.csv",
      sep="," , header=TRUE, stringsAsFactors=FALSE)
acs$Income <- with(acs, FamilyIncome >= 150000)
income1 <- glm(Income ~ HouseCosts + NumWorkers + OwnRent
           + NumBedrooms + FamilyType,data=acs,
           family=binomial(link="logit"))
```

- Logistic Regression

```
summary(income1)
```

```
##
## Call:
## glm(formula = Income ~ HouseCosts + NumWorkers + OwnRent + NumBedroo
##     FamilyType, family = binomial(link = "logit"), data = acs)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.8452  -0.6246  -0.4231  -0.1743   2.9503
##
## Coefficients:
##                          Estimate Std. Error z value Pr(>|z|)
## (Intercept)            -5.738e+00  1.185e-01 -48.421   <2e-16 ***
## HouseCosts              7.398e-04  1.724e-05  42.908   <2e-16 ***
## NumWorkers              5.611e-01  2.588e-02  21.684   <2e-16 ***
## OwnRentOutright         1.772e+00  2.075e-01   8.541   <2e-16 ***
## OwnRentRented          -8.886e-01  1.002e-01  -8.872   <2e-16 ***
## NumBedrooms             2.339e-01  1.683e-02  13.895   <2e-16 ***
## FamilyTypeMale Head     3.336e-01  1.472e-01   2.266   0.0235 *
## FamilyTypeMarried       1.405e+00  8.704e-02  16.143   <2e-16 ***
```

- Logistic Regression

```
invlogit <- function(x)
{
1 / (1 + exp(-x))
}
invlogit(income1$coefficients)
```

```
##          (Intercept)          HouseCosts          NumWorkers      OwnR
##          0.003211572          0.500184950          0.636702036
##       OwnRentRented        NumBedrooms FamilyTypeMale Head   Family
##          0.291408659          0.558200010          0.582624773
```

The Other Models for Data Analysis

1. Nonlinear model

   - Nonlinear Least Squares
   - Splines
   - Generalized Additive Models
   - Decision Trees
   - Random Forests

2. Time Series

   - Autoregressive Moving Average
   - GARCH

3. Clustering

   - K-means
   - Latent Class
   - Hierarchical Clustering

References

```
knitr::include_graphics("Images/R books.png")
```