Chapter 7:
Statistical Modeling

Descriptive Statistics

✓ The following table provides basic statistical functions

Function	Description
mean(x)	Mean
quantile(x, prob)	Quantile at a specified probability
mad(x)	Median absolute deviation
var(x)	Variance
range(x)	Vector containing min and max
cov(x,y)	Covariance
cor(x,y)	Correlation Coefficient

➡ In the above functions, the objects x, y are the operands. They can be applied to almost any data structure, including vector, array, matrix, data frame etc

Descriptive Statistics

- ✓ Mean
  - ➡ Function *mean*() is used to evaluate the mean value of numbers in an object
    - Usage: Type *mean*({object}, …) at the command line
    - Examples:

```
> #Example 1
> x <- c(1,3,5,1,4,5,8,4,3,4,7,4,3,3,3,5,6,3,24,6,3,24,53,3,53,4)
> mean(x)
[1] 9.307692
>
> #Example 2
> dim(x) <- c(2,13)
> x
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13]
[1,]    1    5    4    8    3    7    3    3    6    24     3    53    53
[2,]    3    1    5    4    4    4    3    5    3     6    24     3     4
> mean(x)
[1] 9.307692
>
> #Example 3
> dim(x) <- c(26)
> x
 [1]  1  3  5  1  4  5  8  4  3  4  7  4  3  3  3  5  6  3 24  6  3 24 53  3 53  4
> mean(x)
[1] 9.307692
>
> #Example 4
> x <- as.list(x)
> mean(as.numeric(x[1:26]))
[1] 9.307692
```

Descriptive Statistics

✓ Quantile

➡ Function *quantile*() is used to generate sample quantiles of an object for a given set of probabilities

- Usage: Type *quantile*({object}, prob = , type = , …) at the command line
- Examples:

```
> x <- c(1,3,5,1,4,5,8,4,3,4,7,4,3,3,3,5,6,3,24,6,3,24,53,3,53,4)
> quantile(x,probs=seq(0,1,0.25),type=7)
  0%  25%  50%  75% 100% 
   1    3    4    6   53 
> quantile(x,probs=seq(0,1,0.1),type=7)
  0%  10%  20%  30%  40%  50%  60%  70%  80%  90% 100% 
 1.0  3.0  3.0  3.0  4.0  4.0  5.0  5.5  7.0 24.0 53.0 
> quantile(x,probs=seq(0,1,0.16666),type=7)
     0% 16.666% 33.332% 49.998% 66.664%  83.33% 99.996% 
 1.0000  3.0000  3.0000  4.0000  5.0000  7.8325 53.0000 
> quantile(x,probs=seq(0,1,0.125),type=7)
    0%  12.5%    25%  37.5%    50%  62.5%    75%  87.5%   100% 
 1.000  3.000  3.000  3.375  4.000  5.000  6.000 22.000 53.000 
```

➡ Function *quantile*() will extrapolate values in the object to match the percentage specified by the probabilities to obtain the quantile, if needed

- This particularly applies if there are more quantiles than values in the object

Descriptive Statistics

- ✓ Median Absolute Deviation
  - ➡ Function *mad*() is used to calculate the median absolute deviation
    - Usage: Type *mad*({object}, center = …, constant = , …) at the command line, where *center* is the central value of the data (defaults to the median) and constant is the scaling factor (defaults to 1.4826)
    - Examples:

```
> z <- c(2,3,1,4,6,5,2,4,6,2,3,5,2,4,6,8)
> median(z)
[1] 4
> abs(z-median(z))
 [1] 2 1 3 0 2 1 2 0 2 2 1 1 2 0 2 4
> median(abs(z-median(z)))
[1] 2
> mad(z)
[1] 2.9652
> mad(z,constant=1)
[1] 2
```

Descriptive Statistics

- ✓ Variance
  - ➡ Function *var*() is used to compute the variance of a data set
    - Usage: Type *var*({object}, ...) at the command line
    - Examples:

```
> x <- c(1,4,3,5,2,3,1,3,2,4,2,4,3,5,4,2,2,3,5,2,3,5,6,4,6,7,8,45,6,4,2,4,6,2,4,8,4,7,4,5,2,9)
> x
 [1]  1  4  3  5  2  3  1  3  2  4  2  4  3  5  4  2  2  3  5  2  3  5  6  4  6  7  8 45  6  4  2  4
[33]  6  2  4  8  4  7  4  5  2  9
> sum((x-mean(x))^2)/(length(x)-1)
[1] 43.73113
> var(x)
[1] 43.73113
```

  - ➡ For a data set of *n* values, the denominator is set to *n - 1*, corresponding to the number of degrees of freedom

- ✓ Range
  - ➡ Function *range*() is used to compute the range of a data set, i.e., a vector containing the minimum and maximum values in the data set
    - Usage: Type *range*({object}, ...) at the command line
    - Examples:

```
> x
 [1]  1  4  3  5  2  3  1  3  2  4  2  4  3  5  4  2  2  3  5  2  3  5  6  4  6  7  8 45  6  4  2  4
[33]  6  2  4  8  4  7  4  5  2  9
> c(min(x),max(x))
[1]  1 45
> range(x)
[1]  1 45
```

Descriptive Statistics

- ✓ Covariance
  - ➡ Function *cov*() is used to compute the covariance of two data sets
    - Usage: Type *cov*({object1}, {object2}, use = , method =, …) at the command line. The default *method* is "pearson" and the parameter *use* is optional and specifies how to deal with missing values
    - Examples:

```
> #Example 1
> x <- c(1,2,2,4,2,5,5,7,8,7,9)
> y <- c(9,9,8,8,8,7,5,7,4,2,3)
> cov(x,y)
[1] -5.790909
>
> #Example 2
> #x: same as Example 1
> y <- c(2,2,4,3,4,5,5,6,7,8,8)
> cov(x,y)
[1] 5.472727
```

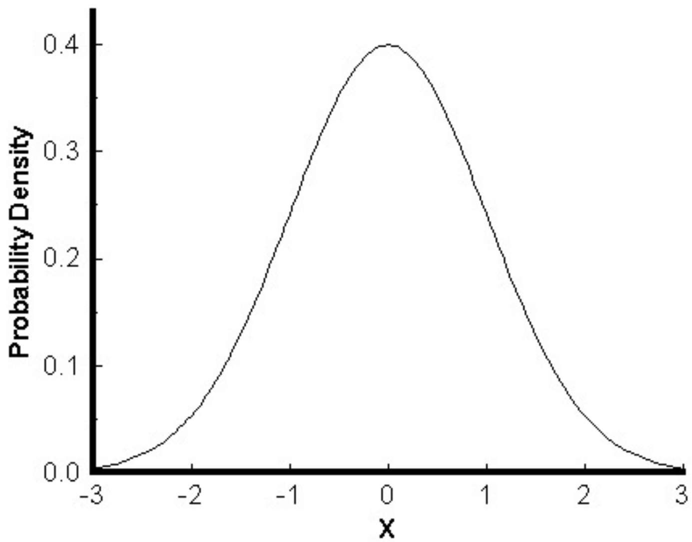
- ✓ Correlation
  - ➡ Function *cor*() is used to compute the correlation between two data sets
    - Usage: Type *cor*({object1}, {object2}, use = , method =, …) at the command line
    - Examples:

```
> #Example 1
> x <- c(1,2,2,4,2,5,5,7,8,7,9)
> y <- c(2,2,4,3,4,5,5,6,7,8,8)
> cor(x,y)
[1] 0.9154816
>
> #Example 2
> #x: same as Example 1
> y <- c(9,9,8,8,8,7,5,7,4,2,3)
> cor(x,y)
[1] -0.8526687
```

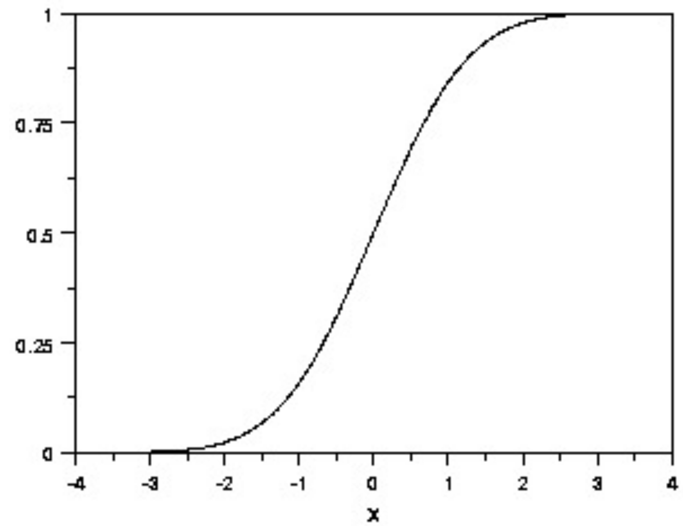
Exercise

Probability Distributions

✓ For a normal distribution ($\mu = 0, \sigma = 1$)



PDF



CDF

Probability Distributions

- ✓ In R, probability distributions are evaluated by the following functions
  - ➡ Function *dxxxx*() provides the PDF (Probability Density Function)
  - ➡ Function *pxxxx*() provides the CDF (Cumulative Distribution Function)
  - ➡ Function *qxxxx*() provides the quantile function
  - ➡ Function *rxxxx*() generates a sample data for a random variable that follows a probability distribution
  - ➡ "*xxxx*" specifies the name of distribution. Ex., norm for Normal, t for Student's T, exp for exponential etc.
- ✓ As long as the data type is numeric - integer, real, these can be applied to any data structures, including vectors, arrays, matrices, data frames, tables etc

Probability Distributions

✓ Normal distribution

➡ Functions *dnorm*(): PDF, *pnorm*(): CDF, *qnorm*(): Quantile, *rnorm*(): simulator

- Usage: Type *dnorm*({object}, mean = , sd = , log =) from the command prompt. Parameter *log* specifies whether or not probabilities (p) are displayed as *log*(p)
- Examples:

```
> x <- c(1,3,4,2,3,4,1,3,4,1,5,6,3,5)
> dnorm(x,mean=10,sd=5)
 [1] 0.01579003 0.02994549 0.03883721 0.02218417 0.02994549 0.03883721 0.01579003 0.02994549
 [9] 0.03883721 0.01579003 0.04839414 0.05793831 0.02994549 0.04839414
> dnorm(x,3,2)
 [1] 0.1209854 0.1994711 0.1760327 0.1760327 0.1994711 0.1760327 0.1209854 0.1994711 0.1760327
[10] 0.1209854 0.1209854 0.0647588 0.1994711 0.1209854
```

- Usage: Type *pnorm*({object}, mean = , sd = , lower.tail = , log.p =) from the command prompt. Parameter *log.p* specifies whether or not probabilities (p) are displayed as *log*(p). Parameter *lower.tail* (default = TRUE) specifies if probabilities are displayed as P[X ≤ x] or P[X > x]
- Examples:

```
> x
 [1] 1 3 4 2 3 4 1 3 4 1 5 6 3 5
> pnorm(x,mean=10,sd=5)
 [1] 0.03593032 0.08075666 0.11506967 0.05479929 0.08075666 0.11506967 0.03593032 0.08075666
 [9] 0.11506967 0.03593032 0.15865525 0.21185540 0.08075666 0.15865525
> pnorm(x,3,2)
 [1] 0.1586553 0.5000000 0.6914625 0.3085375 0.5000000 0.6914625 0.1586553 0.5000000 0.6914625
[10] 0.1586553 0.8413447 0.9331928 0.5000000 0.8413447
```

Probability Distributions

✓ Normal distribution

➡ Functions *dnorm*(): PDF, *pnorm*(): CDF, *qnorm*(): Quantile, *rnorm*(): simulator

- Usage: Type *qnorm*({object}, mean = , sd = , lower.tail = , log.p =) from the command prompt.
- Examples:

```
> x
 [1] 1 3 4 2 3 4 1 3 4 1 5 6 3 5
> y <- pnorm(x,3,2)
> y
 [1] 0.1586553 0.5000000 0.6914625 0.3085375 0.5000000 0.6914625 0.1586553 0.5000000 0.6914625
[10] 0.1586553 0.8413447 0.9331928 0.5000000 0.8413447
> qnorm(y,3,2)
 [1] 1 3 4 2 3 4 1 3 4 1 5 6 3 5
```

- Usage: Type *rnorm*(number, mean = , sd =) from the command prompt.
- Example: Simulating values for a random variable and displaying probabilities

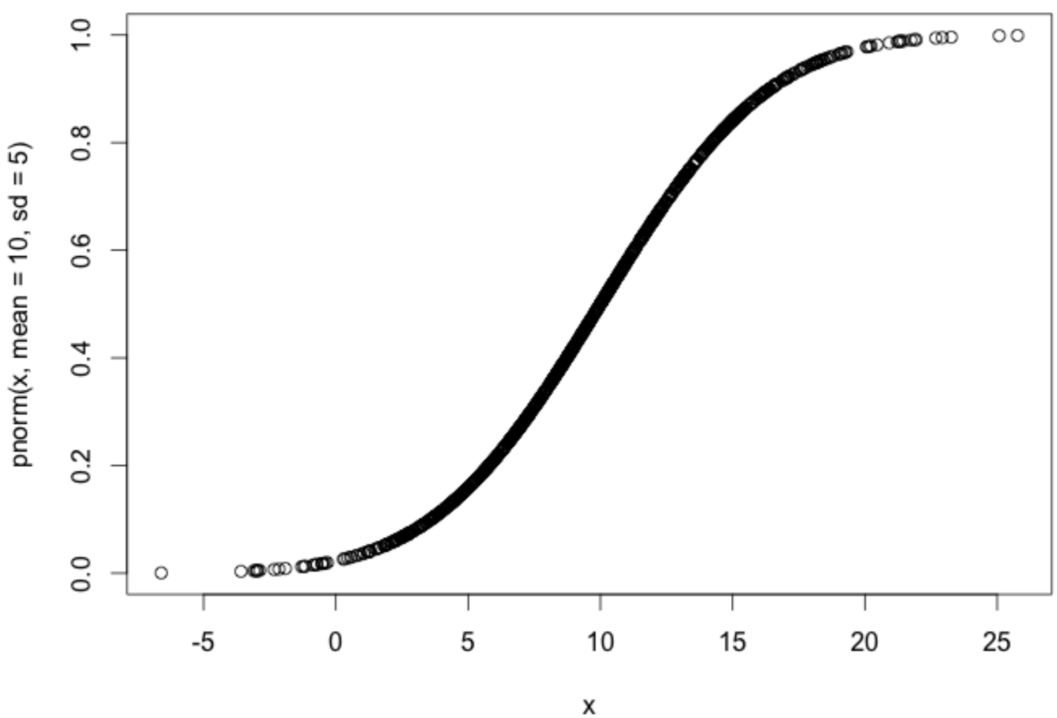
```
> x <- rnorm(5,mean=10,sd=3)
> x
[1] 13.142737  5.187798  4.787276  9.111711  6.234254
> y <- pnorm(x,mean=10,sd=3)
> y
[1] 0.85258372 0.05434961 0.04114290 0.38357824 0.10469427
> z <- list(X = x, Prob = y)
> z <- as.data.frame(z)
> z
          X       Prob
1 13.142737 0.85258372
2  5.187798 0.05434961
3  4.787276 0.04114290
4  9.111711 0.38357824
5  6.234254 0.10469427
```

Probability Distributions

✓ Normal distribution

- Examples: Generate a sample of 1000 values of a normally distributed variable with $\mu = 10$ and $\sigma = 5$. Plot the CDF.

```
> x <- rnorm(1000,mean=10, sd=5)
> x
   [1] 14.4339249  7.6833986  7.6930981 11.6557328  3.8530702 14.0019489 11.9022897  6.5990909
   [9] 10.2084394 20.0683739 16.2537346 13.0851683  6.5778204  8.5538993  7.3791510  6.2723956
  [17] 12.8005906  8.4447949 12.8277080  2.7962086  4.2646677 10.2741557  8.4560572 11.1466960
  [25]  9.1411009  2.3002402  3.9306160 14.7439892 10.8760866  8.6025734  2.7058637 10.2687228
     ⋮
 [969]  6.9656698 16.2231100  8.2488192  0.3387501  4.1581266 13.3591512  7.9529326  3.7498985
 [977] 11.3458365  6.6081132  9.5497666 11.5937697  8.2391186  5.0912407  1.5305825  4.9761633
 [985] 13.8614229  6.7713865  7.0113000  2.5352082 12.2298784  8.3532768  9.4772474 14.5258027
 [993]  5.7751160 18.4313353  9.3177712  3.9688407  3.8757827 13.0691362 19.1461302  9.8903687
> plot(x,pnorm(x,mean=10, sd=5))
```

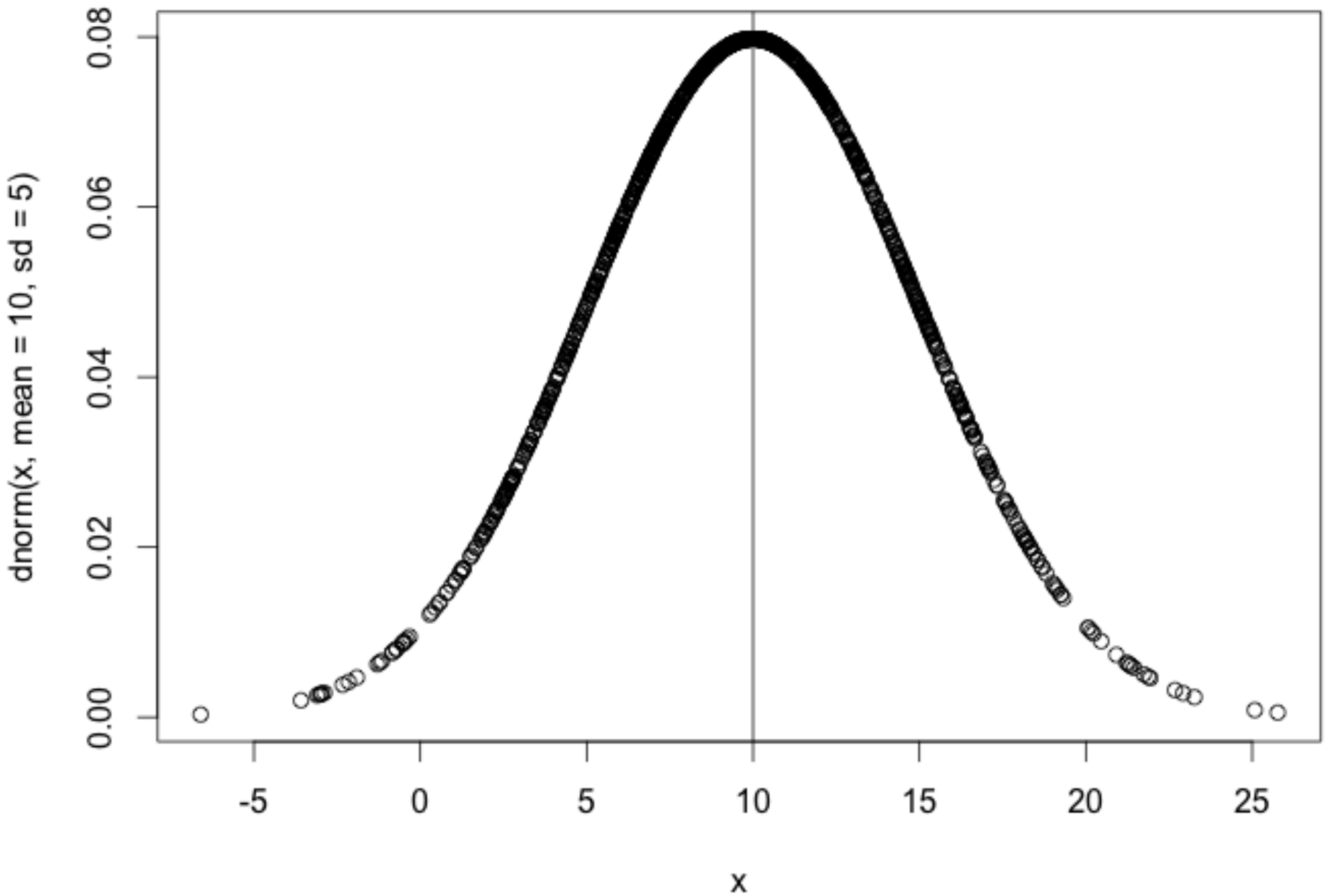


Probability Distributions

✓ Normal distribution

- Examples: Generate a sample of 1000 values of a normally distributed variable with $\mu = 10$ and $\sigma = 5$. Plot the PDF.

```
> plot(x,dnorm(x,mean=10, sd=5))
> a <- c(10,10)
> b <- c(-1,1)
> lines(a,b, type="b")
```



Probability Distributions

✓ Distributions in R - Table:

Distribution	R Name [“	Parameters	Defaults
Beta	beta	shape1, shape2	
Binomial	binom	size, prob	
Cauchy	cauchy	location, scale	0, 1
Chi-squared	chisq	df, ncp	-, 1
Exponential	exp	rate	1
F	f	df1, df2	
Gamma	gamma	shape, rate, scale	-, 1, 1/rate
Geometric	geom	prob	
Lognormal	lnorm	meanlog, sdlog	0, 1
Normal	norm	mean, sd	0, 1
Poisson	pois	Lambda	1
Student’s t	t	df, ncp	0, 1
Uniform	unif	min, max	-, 1
Weibull	weibull	shape, scale	

Exercise

Hypothesis Testing

✓ One sample t-Test

- If H_0 represents the null hypothesis, and H_A represents the alternate hypothesis, a t-Test can be used to either reject or not reject H_0 in favor of H_A
- For a sample data set, let say that H_0: Population mean = μ
- H_A can either be one sided [H_A: Population mean > μ or H_A: Population mean < μ], or two sided [H_A: Population mean ≠ μ]
- Function *t.test*() may be used to test the significance of H_0
  - Usage: Type *t.test*({object}, alternative = , mu = , conf.level = ,…).
    - {object} holds the sample used in the test.
    - "alternative" defines whether it is one or two sided (default).
    - "mu" is used to specify the true value of the population parameter and
    - "conf.level" specifies the confidence level of the test. Default value = 0.95
  - Example: Test for significance of H_0: Population mean = 3.3 (two-sided)

```
> x
 [1] 1 2 4 1 3 1 2 3 4 2 4 5 6 4 7 6 2 3 1 3 4 5 5 3 3 1 3 4 5 6 2 1
> x <- c(1,2,3,1,2,1,3,4,2,1,3,5,6,3,2,5,4,5,3,5,6,3,2,3,4,1,2,6,5,2,3,1,4,5,3)
> t.test(x, mu=3.3, alternative="two.sided",conf.level=0.95)

 One Sample t-test

data:  x
t = -0.4812, df = 34, p-value = 0.6335
alternative hypothesis: true mean is not equal to 3.3
95 percent confidence interval:
 2.628380 3.714477
sample estimates:
mean of x
 3.171429
```

Hypothesis Testing

✓ One sample t-Test

- Example: Test for significance of H_0: Population mean = 3.3. H_A: Mean > 3.3

```
> t.test(x, mu=3.3, alternative="greater",conf.level=0.95)

 One Sample t-test

data:  x
t = -0.4812, df = 34, p-value = 0.6833
alternative hypothesis: true mean is greater than 3.3
95 percent confidence interval:
 2.719586      Inf
sample estimates:
mean of x
 3.171429
```

- Example: Test for significance of H_0: Population mean = 3.9. H_A: Mean < 3.9

```
> t.test(x, mu = 3.9, alternative="less",conf.level=0.95)

 One Sample t-test

data:  x
t = -2.7265, df = 34, p-value = 0.005023
alternative hypothesis: true mean is less than 3.9
95 percent confidence interval:
     -Inf 3.623271
sample estimates:
mean of x
 3.171429
```

Hypothesis Testing

✓ One sample t-Test

- Example: Test for significance of H_0: Population mean = 3.7 [two-sided]

```
> y <- t.test(x, mu = 3.7, alternative="two.sided",conf.level=0.95)
> y

 One Sample t-test

data:  x
t = -1.9781, df = 34, p-value = 0.05607
alternative hypothesis: true mean is not equal to 3.7
95 percent confidence interval:
 2.628380 3.714477
sample estimates:
mean of x
 3.171429

> typeof(y)
[1] "list"
> class(y)
[1] "htest"
> names(y)
[1] "statistic"   "parameter"   "p.value"     "conf.int"    "estimate"    "null.value"  "alternative"
[8] "method"      "data.name"
> attributes(y)
$names
[1] "statistic"   "parameter"   "p.value"     "conf.int"    "estimate"    "null.value"  "alternative"
[8] "method"      "data.name"

$class
[1] "htest"

> y$statistic
        t
-1.978066
> y$parameter
df
34
> y$p.value
[1] 0.05607098
```

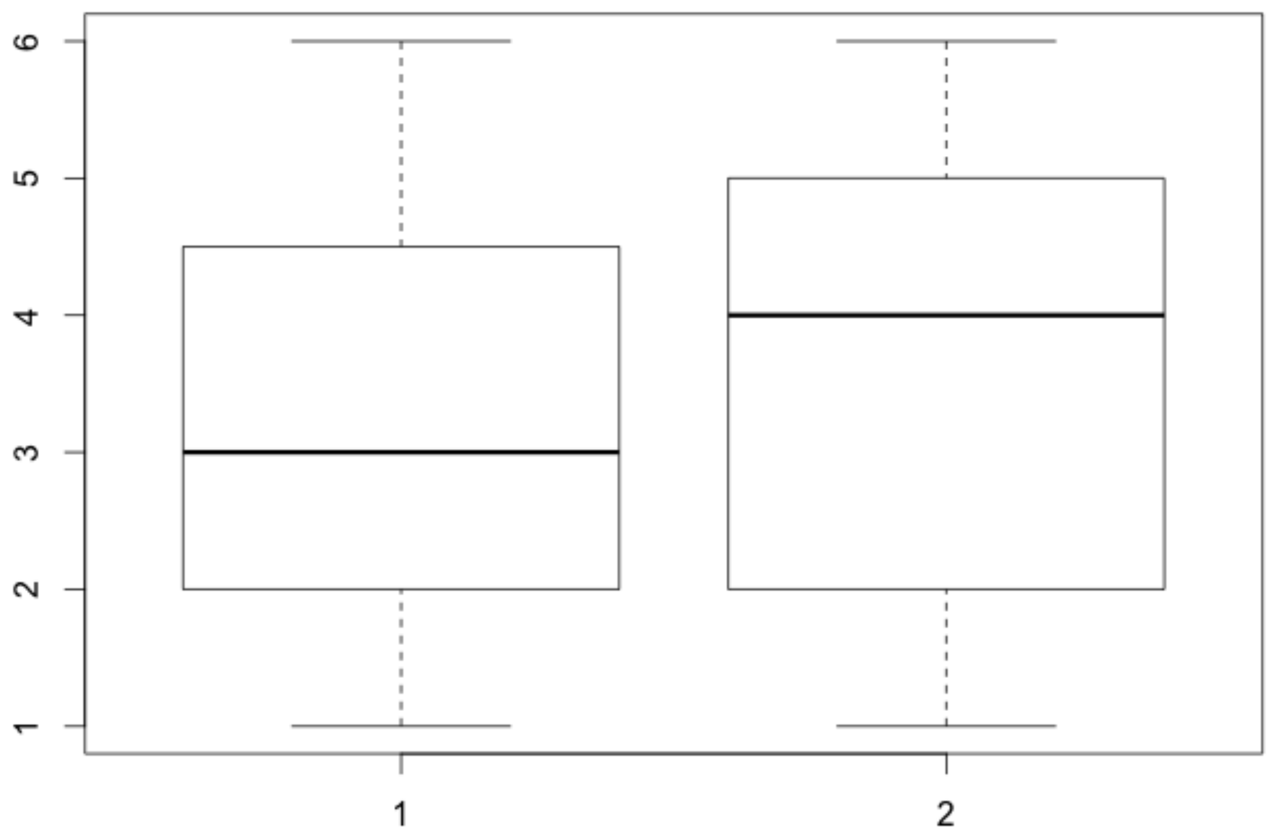
Hypothesis Testing

- ✓ Two sample t-Test
  - ➡ If H_0 represents the null hypothesis, and H_A represents the alternate hypothesis, a t-Test can be used to either reject or not reject H_0 in favor of H_A
  - ➡ If there are two sample data sets, then the question becomes whether these samples have been drawn from the same population, i.e., whether they have the same mean/standard deviation
  - ➡ In this case, let say that H_0: $\mu 1 - \mu 2 = 0$, which means H_A: $\mu 1 - \mu 2 \neq 0$
  - ➡ Function *t.test*() may be used to test the significance of H_0
    - Usage: Type *t.test*({object1}, {object2}, alternative = , mu = , conf.level = ,…).
      - {object1} holds the 1st sample and {object2} the 2nd
      - "alternative" defines whether it is one or two sided (default).
      - "mu" still specifies the true value of the population parameter. In this case, the parameter is $\mu 1 - \mu 2$; thus "mu" may be set to the default value of 0
      - "conf.level" specifies the confidence level of the test. Default value = 0.95

Hypothesis Testing

- Two sample t-Test
  - Example: Two sample data sets x and y

```
> x <- c(1,2,3,1,2,1,3,4,2,1,3,5,6,3,2,5,4,5,3,5,6,3,2,3,4,1,2,6,5,2,3,1,4,5,3)
> x
 [1] 1 2 3 1 2 1 3 4 2 1 3 5 6 3 2 5 4 5 3 5 6 3 2 3 4 1 2 6 5 2 3 1 4 5 3
> y <- c(1,3,3,5,2,1,2,4,5,6,3,2,4,5,6,3,2,4,5,6,2,1,5,2,4,5,6,4,6,5,3,1,5,6,3)
> y
 [1] 1 3 3 5 2 1 2 4 5 6 3 2 4 5 6 3 2 4 5 6 2 1 5 2 4 5 6 4 6 5 3 1 5 6 3
> median(x)
[1] 3
> median(y)
[1] 4
> boxplot(x,y)
```



Hypothesis Testing

✓ Two sample t-Test

- Example: Test for significance of H_0: $\mu 1 - \mu 2 = 0$

```
> x
 [1] 1 2 3 1 2 1 3 4 2 1 3 5 6 3 2 5 4 5 3 5 6 3 2 3 4 1 2 6 5 2 3 1 4 5 3
> y <- c(1,3,3,5,2,1,2,4,5,6,3,2,4,5,6,3,2,4,5,6,2,1,5,2,4,5,6,4,6,5,3,1,5,6,3)
> t.test(x,y)

 Welch Two Sample t-test

data:  x and y
t = -1.3954, df = 67.784, p-value = 0.1675
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -1.3192305  0.2335163
sample estimates:
mean of x mean of y
 3.171429  3.714286
```

- Example: If $\mu 1 - \mu 2 \neq 0$, then test for significance of H_0: $\mu 1 - \mu 2 = 0.5$

```
> t.test(x,y,m=0.5)

 Welch Two Sample t-test

data:  x and y
t = -2.6806, df = 67.784, p-value = 0.00922
alternative hypothesis: true difference in means is not equal to 0.5
95 percent confidence interval:
 -1.3192305  0.2335163
sample estimates:
mean of x mean of y
 3.171429  3.714286
```

Exercise

Hypothesis Testing

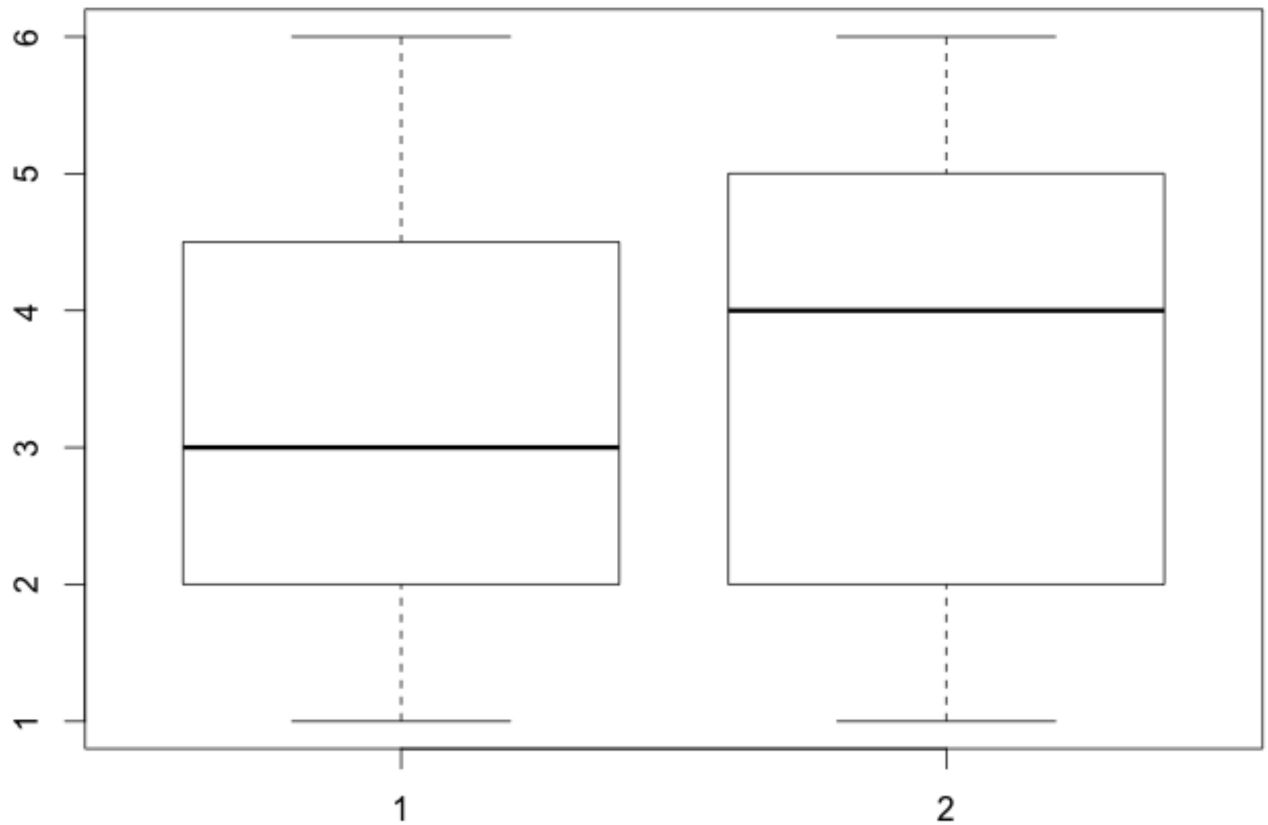
- ✓ KS Test
  - ➡ If H_0 represents the null hypothesis, and H_A represents the alternate hypothesis, a KS-Test can be used to either reject or not reject H_0 in favor of H_A
  - ➡ If there are two sample data sets, then the question becomes whether these samples have been drawn from the same population, i.e., whether they have the same mean/standard deviation
  - ➡ In this case, let say that H_0: data sets do not vary significantly, which means H_A: data sets vary significantly
  - ➡ Like *t.test*(), function *ks.test*() may be used to test the significance of H_0 using the Kolmogrov-Smirnov test
    - Usage: Type *ks.test*({object1}, {object2}, alternative = , exact = …).
      - {object1} holds the 1st sample and {object2} the 2nd
      - "alternative" defines whether it is one or two sided (default).
      - "exact" specifies whether an exact p-value should be computed. Exact p-values are not available if the test is one-sided or in the presence of ties.

Hypothesis Testing

✓ KS Test

- Example: Two sample data sets x and y

```
> x <- c(1,2,3,1,2,1,3,4,2,1,3,5,6,3,2,5,4,5,3,5,6,3,2,3,4,1,2,6,5,2,3,1,4,5,3)
> x
 [1] 1 2 3 1 2 1 3 4 2 1 3 5 6 3 2 5 4 5 3 5 6 3 2 3 4 1 2 6 5 2 3 1 4 5 3
> y <- c(1,3,3,5,2,1,2,4,5,6,3,2,4,5,6,3,2,4,5,6,2,1,5,2,4,5,6,4,6,5,3,1,5,6,3)
> y
 [1] 1 3 3 5 2 1 2 4 5 6 3 2 4 5 6 3 2 4 5 6 2 1 5 2 4 5 6 4 6 5 3 1 5 6 3
> median(x)
[1] 3
> median(y)
[1] 4
> boxplot(x,y)
```



Hypothesis Testing

- **KS Test**
  - Example: Test for significance of H_0

```
> x
 [1] 1 2 3 1 2 1 3 4 2 1 3 5 6 3 2 5 4 5 3 5 6 3 2 3 4 1 2 6 5 2 3 1 4 5 3
> y
 [1] 1 3 3 5 2 1 2 4 5 6 3 2 4 5 6 3 2 4 5 6 2 1 5 2 4 5 6 4 6 5 3 1 5 6 3
> ks.test(x,y)

 Two-sample Kolmogorov-Smirnov test

data:  x and y
D = 0.1714, p-value = 0.6826
alternative hypothesis: two-sided

Warning message:
In ks.test(x, y) : cannot compute exact p-values with ties
```

Hypothesis Testing

- ✓ F test to compare two variances
  - ➡ If H_0 represents the null hypothesis, and H_A represents the alternate hypothesis, a t-Test can be used to either reject or not reject H_0 in favor of H_A
  - ➡ If there are two sample data sets, then the question becomes whether these samples have been drawn from the same population, i.e., whether they have the same mean/standard deviation
  - ➡ In this case, let say that H_0: $\sigma1/\sigma2 = 1$, which means H_A: $\sigma1/\sigma2 \neq 1$
  - ➡ Function *var.test*() may be used to test the significance of H_0
    - Usage: Type *var.test*({object1}, {object2}, alternative = , ratio = , conf.level = ,…).
      - {object1} holds the 1st sample and {object2} the 2nd
      - "alternative" defines whether it is one or two sided (default).
      - "ratio" specifies the true value of the population parameter. In this case, the parameter is $\sigma1/\sigma2$; thus "ratio" may be set to the default value of 1
      - "conf.level" specifies the confidence level of the test. Default value = 0.95

Hypothesis Testing

✓ F test to compare two variances

- Example: Test for significance of H_0: $\sigma 1/\sigma 2 = 1$

```
> var.test(x,y)

 F test to compare two variances

data:  x and y
F = 0.8931, num df = 34, denom df = 34, p-value = 0.7436
alternative hypothesis: true ratio of variances is not equal to 1
95 percent confidence interval:
 0.4508023 1.7693239
sample estimates:
ratio of variances
         0.8930931
```

- Example: If $\sigma 1/\sigma 2 \neq 1$, then test for significance of H_0: $\sigma 1/\sigma 2 = 2$

```
> var.test(x,y,ratio=2)

 F test to compare two variances

data:  x and y
F = 0.4465, num df = 34, denom df = 34, p-value = 0.02128
alternative hypothesis: true ratio of variances is not equal to 2
95 percent confidence interval:
 0.4508023 1.7693239
sample estimates:
ratio of variances
         0.8930931
```

Hypothesis Testing

✓ Other tests

R Name	Purpose
chisq.test	Pearson Chi-Square goodness of Fit Test
shapiro.test	Shapiro-Wilk test of normality
wilcox.test	One or two sample Wilcoxon tests

Exercise

Linear Models

✓ Linear models

- ➡ Lets take a simple case of linear regression of two independent variables. The mathematical equation is

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \varepsilon$$

- ➡ In R, this is usually modeled by the ~ sign and assigned to a *formula* object

$$Y <- y \sim x1 + x2$$

- ➡ Y is a *formula* object. It can now be used in a modeling function; any data set representing y, x1 and x2 can be passed to this object to perform the above regression
- ➡ In general, formula objects are defined in R as

$$\text{response} \sim \text{expression}$$

- Example:

```
> Y <- y~x1+x2
> Y
y ~ x1 + x2
> typeof(Y)
[1] "language"
> class(Y)
[1] "formula"
```

Linear Models

✓ Operators

- ➡ Symbols -, *, ^, : and / take on a different meaning as opposed to subtraction, multiplication, exponentiation etc

- ➡ The : operator is used to model an interaction between variables. For example, the expression

  $$y \sim x1 + x2 + x1{:}x2$$

  corresponds to

  $$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_{12} x_1 x_2 + \varepsilon$$

- ➡ The * operator is used to model all combinations of variables including interactions. In the above example, y ~ x1 + x2 + x1:x2 can be shortened as

  $$y \sim x1*x2$$

  - Example:

```
> Y <- y~x1*x2*x3
> terms(Y)
y ~ x1 * x2 * x3
attr(,"term.labels")
[1] "x1"       "x2"       "x3"       "x1:x2"    "x1:x3"    "x2:x3"    "x1:x2:x3"
```

Linear Models

- Operators
  - The ^ operator is used to model interactions up to a certain order. For example, the expression

$$y \sim (x1 + x2 + x3)\text{^}2$$

is equivalent to

$$y \sim x1 + x2 + x3 + x1{:}x2 + x1{:}x3 + x2{:}x3$$

    - Example:

```
> Y <- y ~ (x1 + x2 + x3)^2
> terms(Y)
y ~ (x1 + x2 + x3)^2
attr(,"term.labels")
[1] "x1"    "x2"    "x3"    "x1:x2" "x1:x3" "x2:x3"
```

  - The - operator is used to leave out terms in a formula. For example, the expression

$$y \sim x1*x2*x3 - x1{:}x2{:}x3$$

is equivalent to

$$y \sim x1 + x2 + x3 + x1{:}x2 + x1{:}x3 + x2{:}x3$$

    - Example:

```
> Y <- y ~ (x1*x2*x3)-x1:x2:x3
> terms(Y)
y ~ (x1 * x2 * x3) - x1:x2:x3
attr(,"term.labels")
[1] "x1"    "x2"    "x3"    "x1:x2" "x1:x3" "x2:x3"
```

Linear Models

✓ Operators

➡ The / operator is used when factors are involved. If it involves a factor and a variable, it is used to perform separate regression models with levels of the factor. For example, the expression

y ~ A/x

produces different regression models of y on x within the levels of A

➡ The function *I*() is used to maintain the identity of a term in an formula. For instance, use

y ~ x1 + *I*(x2^m), where m is an exponent of x2

- All operators have their normal meaning when used with *I*()
- Example:

```
> Y <- y ~ x1 + I(x2^3)
> terms(Y)
y ~ x1 + I(x2^3)
attr(,"term.labels")
[1] "x1"      "I(x2^3)"
```

Linear Models

✓ Examples

y ~ x1, y ~ 1 + x1	Simple linear regression of y with x1
y ~ 0 + x1, y ~ -1 + x1	Simple linear regression of y with x1 without the intercept
y ~ x1 + x2 + x3	Multiple linear regression involving 3 variables
log(y) ~ x1 + x2	Regression of y with x1 and x2, involving a Log Transformation of y
y ~ sqrt(x1) + x2	Regression of y with the square root of x1, and x2
y ~ A	Classification analysis of y with classes determined by factor A
y ~ A + x1	Classification analysis of y, with one factor and a variable
y ~ poly(x,2), y ~ 1 + x + I(x^2)	Polynomial regression up to power 2
y ~ A*B, y ~ A + B + A:B	Cross classification of two factors
y ~ B %in% A	Nested classification of B in A. Similar to B:A
y ~ A/B	Equivalent to y ~ A + B %in% A
exp(y) ~ x1*x2*x3	Regression of y with x1 and x2, involving a Exponential Transformation of y

Exercise

Linear Models

- ✓ Generating a linear model
  - ➡ Function *lm*() may be used to generate a linear model
    - Usage: Type *lm*({formula}, data = , weights = , subset =) from the command prompt.
      - Parameter *weights* is to specify an optional vector of weights to be used in the least squares model.
      - Parameter *subset* is an optional vector specifying a subset of data to be fit.
    - Example: In "datasets", create a linear model of mileage as a function of displacement, horsepower, weight and # cylinders

```
> mtcars
                mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Mazda RX4      21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag  21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
Datsun 710     22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
    :
Ferrari Dino   19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
Maserati Bora  15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
Volvo 142E     21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2

> MTCARSFORM <- mtcars$mpg ~ mtcars$cyl + mtcars$disp + mtcars$hp + mtcars$wt
> mtcars.lm <- lm(MTCARSFORM,data=mtcars)
> mtcars.lm

Call:
lm(formula = MTCARSFORM, data = mtcars)

Coefficients:
(Intercept)   mtcars$cyl  mtcars$disp    mtcars$hp    mtcars$wt
   40.82854     -1.29332      0.01160     -0.02054     -3.85390
```

Linear Models

✓ Mining the model

- Example: In the prior example, obtain details associated with mtcars.lm

```
> names(mtcars.lm)
 [1] "coefficients"  "residuals"     "effects"       "rank"          "fitted.values" "assign"
 [7] "qr"            "df.residual"   "xlevels"       "call"          "terms"         "model"
> mtcars.lm$coefficients
(Intercept)  mtcars$cyl mtcars$disp   mtcars$hp   mtcars$wt
40.82853674 -1.29331972  0.01159924 -0.02053838 -3.85390352
> mtcars.lm$residuals
          Mazda RX4       Mazda RX4 Wag          Datsun 710      Hornet 4 Drive   Hornet Sportabout
        -1.56804806         -0.58530266         -3.25685052         -0.01170091          0.89393888
            Valiant          Duster 360           Merc 240D            Merc 230            Merc 280
        -2.08741154         -1.56736731          0.61046532         -0.39748889         -0.02900250
             :
      Maserati Bora          Volvo 142E
         1.66544168         -2.70623099
> mtcars.lm$qr
$qr
                    (Intercept)   mtcars$cyl   mtcars$disp     mtcars$hp    mtcars$wt
Mazda RX4            -5.6568542 -35.00178567 -1.305160e+03 -8.297898e+02 -18.199514334
Mazda RX4 Wag         0.1767767   9.94359090  6.224581e+02  3.177801e+02   4.262896616
Datsun 710            0.1767767   0.21715832 -2.978770e+02 -3.542113e+01  -2.298719924
     :
Maserati Bora         0.1767767  -0.18511084 -1.458785e-01 -6.634167e-01  -0.002901978
Volvo 142E            0.1767767   0.21715832  1.570687e-01 -1.615312e-01  -0.116007083
attr(,"assign")
[1] 0 1 2 3 4

$qraux
[1] 1.176777 1.016024 1.113427 1.166614 1.258994

$pivot
[1] 1 2 3 4 5
    :
```

Linear Models

- ✓ Mining the model
  - Example: In the prior example, obtain details associated with mtcars.lm

```
> names(mtcars.lm)
 [1] "coefficients"  "residuals"     "effects"       "rank"          "fitted.values" "assign"
 [7] "qr"            "df.residual"   "xlevels"       "call"          "terms"         "model"
> mtcars.lm$rank
[1] 5
> mtcars.lm$effects
  (Intercept)    mtcars$cyl   mtcars$disp     mtcars$hp     mtcars$wt
-113.64973741  -28.59568066    6.13139078   -3.06119736   -9.53545950   -1.70125977   -1.32680880

   1.68810228    0.72759096    0.07514672   -1.32485328    1.85349371    1.32145872   -0.56794790

   0.25066268    1.28866628    5.72544642    6.01359650    1.23937626    5.99771380   -3.28845154

  -2.31505957   -2.89040015   -1.13055585    2.85529864   -0.20431988   -0.29905762    2.12432052

  -1.01327965   -0.89509751    1.83010027   -1.78589138
> mtcars.lm$fitted.values
          Mazda RX4       Mazda RX4 Wag          Datsun 710      Hornet 4 Drive   Hornet Sportabout
           22.56805            21.58530            26.05685            21.41170            17.80606
            Valiant          Duster 360           Merc 240D            Merc 230            Merc 280
           20.18741            15.86737            23.78953            23.19749            19.22900
              :
> mtcars.lm$assign
[1] 0 1 2 3 4
> mtcars.lm$df.residual
[1] 27
> mtcars.lm$call
lm(formula = MTCARSFORM, data = mtcars)
> mtcars.lm$xlevels
named list()
```

Linear Models

- ✓ Mining the model
  - ➡ Function *summary*() may be used to obtain a summary of the model
    - Usage: Type *summary*({model}) from the command prompt.
    - Example: Summary for mtcars.lm

```
> summary(mtcars.lm)

Call:
lm(formula = MTCARSFORM, data = mtcars)

Residuals:
    Min      1Q  Median      3Q     Max 
-4.0562 -1.4636 -0.4281  1.2854  5.8269 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) 40.82854    2.75747  14.807 1.76e-14 ***
mtcars$cyl  -1.29332    0.65588  -1.972 0.058947 .  
mtcars$disp  0.01160    0.01173   0.989 0.331386    
mtcars$hp   -0.02054    0.01215  -1.691 0.102379    
mtcars$wt   -3.85390    1.01547  -3.795 0.000759 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1 

Residual standard error: 2.513 on 27 degrees of freedom
Multiple R-squared: 0.8486, Adjusted R-squared: 0.8262 
F-statistic: 37.84 on 4 and 27 DF,  p-value: 1.061e-10
```

Linear Models

- ✓ Mining the model
  - ➡ Functions *coef*(), *resid*(), *fitted*() may be used to obtain the co-efficients, residuals, and fitted values respectively of a model
    - Usage: Type *coef*({model}) or *resid*({model}) or *fitted*({model}) from the command prompt.
    - Results are identical to {model}$coefficients, {model}$residuals and {model}$fitted.values respectively
    - Example: For mtcars.lm

```
> coef(mtcars.lm)
(Intercept)  mtcars$cyl mtcars$disp   mtcars$hp   mtcars$wt 
40.82853674 -1.29331972  0.01159924 -0.02053838 -3.85390352 
> mtcars.lm$coefficients
(Intercept)  mtcars$cyl mtcars$disp   mtcars$hp   mtcars$wt 
40.82853674 -1.29331972  0.01159924 -0.02053838 -3.85390352 

> resid(mtcars.lm)
          Mazda RX4       Mazda RX4 Wag          Datsun 710      Hornet 4 Drive   Hornet Sportabout 
        -1.56804806         -0.58530266         -3.25685052         -0.01170091          0.89393888 
            Valiant          Duster 360           Merc 240D            Merc 230            Merc 280 
        -2.08741154         -1.56736731          0.61046532         -0.39748889         -0.02900250 
                  :
> mtcars.lm$residuals
          Mazda RX4       Mazda RX4 Wag          Datsun 710      Hornet 4 Drive   Hornet Sportabout 
        -1.56804806         -0.58530266         -3.25685052         -0.01170091          0.89393888 
            Valiant          Duster 360           Merc 240D            Merc 230            Merc 280 
        -2.08741154         -1.56736731          0.61046532         -0.39748889         -0.02900250 
```

Linear Models

✓ Analysis of Variance

➡ Function *anova*() may be used to compute ANOVA tables for one or more models

- Usage: Type *anova*({object1}, …) from the command prompt.
- Example: ANOVA table for mtcars.lm

```
> anova(mtcars.lm)
Analysis of Variance Table

Response: mtcars$mpg
            Df Sum Sq Mean Sq  F value    Pr(>F)
mtcars$cyl   1 817.71  817.71 129.5335 8.251e-12 ***
mtcars$disp  1  37.59   37.59   5.9552 0.0215174 *
mtcars$hp    1   9.37    9.37   1.4844 0.2336201
mtcars$wt    1  90.92   90.92  14.4034 0.0007589 ***
Residuals   27 170.44    6.31
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

- Example: Comparing two different models for mileage based on displacement, weight, horsepower and number of cylinders

```
> formula(mtcars.lm)
mtcars$mpg ~ mtcars$cyl + mtcars$disp + mtcars$hp + mtcars$wt
> formula(mtcars.lm1)
mtcars$mpg ~ mtcars$cyl + mtcars$disp + mtcars$hp + I(1/mtcars$wt)
> anova(mtcars.lm, mtcars.lm1)
Analysis of Variance Table

Model 1: mtcars$mpg ~ mtcars$cyl + mtcars$disp + mtcars$hp + mtcars$wt
Model 2: mtcars$mpg ~ mtcars$cyl + mtcars$disp + mtcars$hp + I(1/mtcars$wt)
  Res.Df    RSS Df Sum of Sq F Pr(>F)
1     27 170.44
2     27 137.13  0    33.318
```

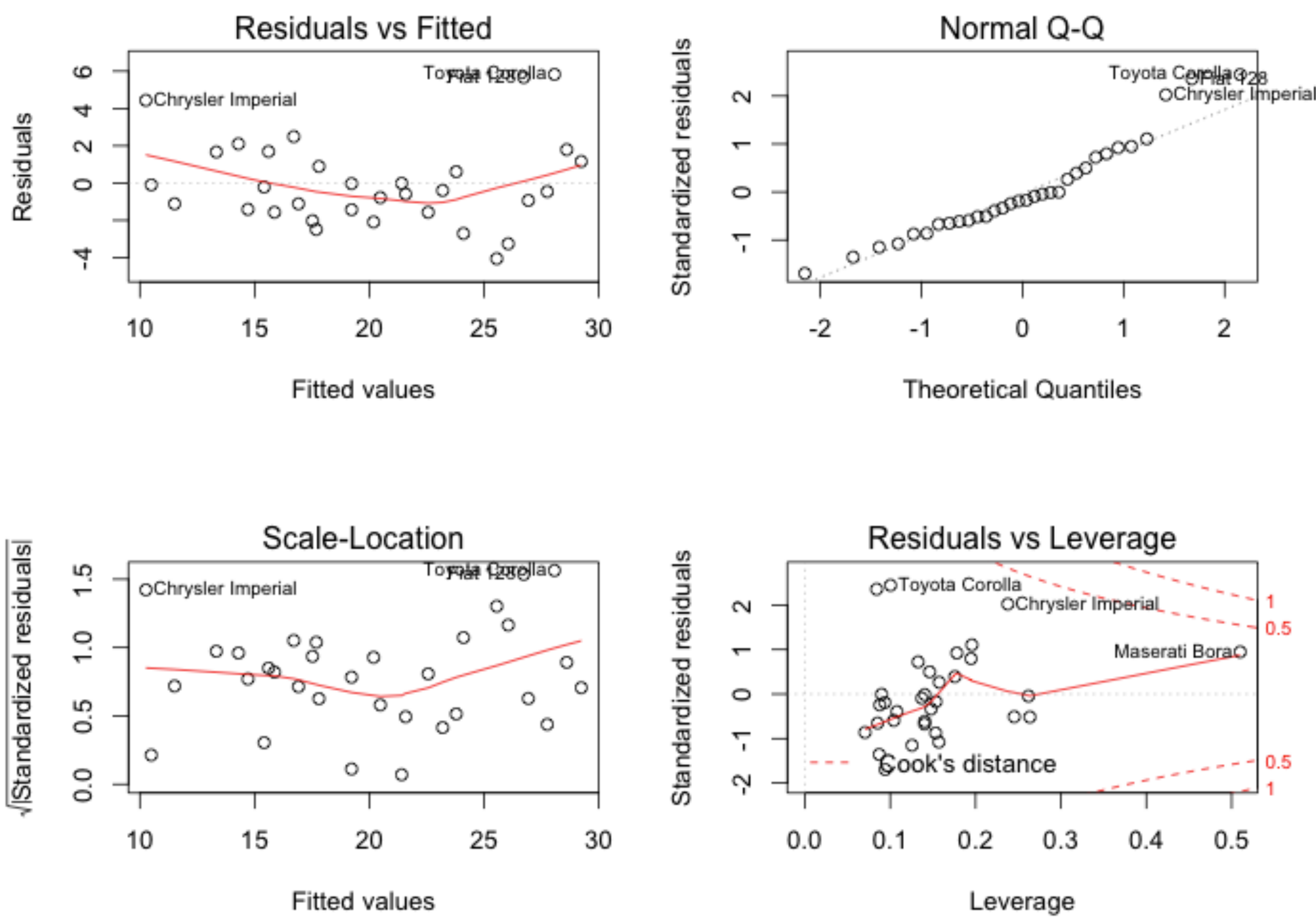
Linear Models

✓ Graphing the model

➡ Function *plot*() allows for a graphical representation of the results

- Usage: Type *plot*({model}) from the command prompt.
- Example: For mtcars.lm

```
> par(mfrow=c(2,2))
> plot(mtcars.lm)
```



Linear Models

✓ Obtaining information

➡ Other functions:

Function	Purpose
deviance	Returns the deviance of a fitted model
predict	Generic function for predictions from fitted models
formula	Generic function for extracting formulae
step	Choose a model in a stepwise algorithm
proj	Returns a matrix of projections of data onto the model terms
family	Specifies the family of models
kappa	Estimates the condition number

Exercise

Linear Models

✓ Updating a model

➡ Function *add1*() may be used to simulate adding a term to a model formula

- Usage: Type *add1*({model}, {new term}) from the command prompt.
- Example: Add data for flow rate [qsec] to mtcars.lm

```
> add1(mtcars.lm,.~.+mtcars$qsec)
Single term additions

Model:
mtcars$mpg ~ mtcars$cyl + mtcars$disp + mtcars$hp + mtcars$wt
            Df Sum of Sq    RSS    AIC
<none>                   170.44 63.526
mtcars$qsec  1     1.759 168.69 65.194
```

➡ Function *drop1*() may be used to understand the effects of dropping a term from a model formula

- Usage: Type *drop1*({model}) from the command prompt.
- Example: Drop terms from mtcars.lm

```
> drop1(mtcars.lm)
Single term deletions

Model:
mtcars$mpg ~ mtcars$disp + mtcars$cyl + mtcars$hp + mtcars$wt
            Df Sum of Sq    RSS    AIC
<none>                   170.44 63.526
mtcars$disp  1     6.176 176.62 62.665
mtcars$cyl   1    24.546 194.99 65.831
mtcars$hp    1    18.048 188.49 64.746
mtcars$wt    1    90.925 261.37 75.206
```

Linear Models

- ✓ Updating a model
  - ➡ Function *update*() may be used to change the formula of a model
    - Usage: Type *update*({model}, {changes}) from the command prompt.
    - Example: Add data for flow rate [qsec] to mtcars.lm

```
> mtcars1.lm <- update(mtcars.lm,.~.+mtcars$qsec)
> summary(mtcars1.lm)

Call:
lm(formula = mtcars$mpg ~ mtcars$cyl + mtcars$disp + mtcars$hp +
    mtcars$wt + mtcars$qsec, data = mtcars)

Residuals:
    Min      1Q  Median      3Q     Max
-4.3117 -1.3483 -0.4352  1.2603  5.6094

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 35.87361    9.91809   3.617  0.00126 **
mtcars$cyl  -1.15608    0.71525  -1.616  0.11809
mtcars$disp  0.01195    0.01191   1.004  0.32484
mtcars$hp   -0.01584    0.01527  -1.037  0.30908
mtcars$wt   -4.22527    1.25239  -3.374  0.00233 **
mtcars$qsec  0.25382    0.48746   0.521  0.60699
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.547 on 26 degrees of freedom
Multiple R-squared: 0.8502, Adjusted R-squared: 0.8214
F-statistic: 29.51 on 5 and 26 DF,  p-value: 6.182e-10
```

Exercise

Generalized Linear Models

✓ Introduction

➡ Linear modeling in R has been developed further to accommodate non-normal response distributions and transformations to linearity.

- Example: Logistic Regression where the binomial distribution comes into play

➡ Function *glm*() may be used to generate a linear model

- Usage: Type *glm*({formula}, family = , data = , weights = , subset =) from the command prompt.
  - Parameter *family* is to specify either the response distribution or use quasi-likelihood models where the distribution is not specified
  - Parameter *weights* is to specify an optional vector of weights to be used in the least squares model.
  - Parameter *subset* is an optional vector specifying a subset of data to be fit.
- The interpretation of a formula object is the same for a *glm* compared to an *lm*; all the rules associated with defining an *lm* along with the use of operators also applies to a *glm*

➡ Table of families and their link functions

Family	Link Function
binomial	logit, probit, log, cloglog
gaussian	identity, log, inverse
Gamma	identity, log, inverse
inverse.gaussian	1/mu^2, identity, log, inverse
poisson	log, identity, sqrt
quasi	logit, probit, cloglog, identity, inverse, log, 1/mu^2 and sqrt

Generalized Linear Models

✓ Generalized linear models

- Example: Perform a logistic regression on data on Low Birth Weights and the relationship to the age, weight during last menstrual period and number of physician visits during the first trimester

```
> LBwt <- read.table(file="lowbwt.dat",skip=4,header=TRUE,sep="")
> LBwt
     ID LOW AGE LWT RACE SMOKE PTL HT UI FTV  BWT
1    85   0  19 182    2     0   0  0  1   0 2523
2    86   0  33 155    3     0   0  0  0   3 2551
3    87   0  20 105    1     1   0  0  0   1 2557
      :
187  82   1  23  94    3     1   0  0  0   0 2495
188  83   1  17 142    2     0   0  1  0   0 2495
189  84   1  21 130    1     1   0  1  0   3 2495
> LBForm <- LOW ~ AGE + LWT + FTV
> LBwt.glm <- glm(LBForm,family=binomial,data=LBwt)
> LBwt.glm

Call:  glm(formula = LBForm, family = binomial, data = LBwt)

Coefficients:
(Intercept)          AGE          LWT          FTV
    1.72065     -0.03746     -0.01262     -0.05844

Degrees of Freedom: 188 Total (i.e. Null);  185 Residual
Null Deviance:      234.7
Residual Deviance: 227  AIC: 235
```

Generalized Linear Models

✓ Mining the model

- Example: From LBwt.glm, obtain the summary

```
> summary(LBwt.glm)

Call:
glm(formula = LBForm, family = binomial, data = LBwt)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.1443  -0.9066  -0.7487   1.3413   2.0255

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)  1.720652   0.999976   1.721   0.0853 .
AGE         -0.037461   0.032955  -1.137   0.2557
LWT         -0.012624   0.006231  -2.026   0.0427 *
FTV         -0.058438   0.166131  -0.352   0.7250
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 234.67  on 188  degrees of freedom
Residual deviance: 227.00  on 185  degrees of freedom
AIC: 235

Number of Fisher Scoring iterations: 4
```

Generalized Linear Models

✓ Mining the model

- Example: From LBwt.glm, obtain some attributes and their values

```
> names(LBwt.glm)
 [1] "coefficients"      "residuals"         "fitted.values"     "effects"
 [5] "R"                 "rank"              "qr"                "family"
 [9] "linear.predictors" "deviance"          "aic"               "null.deviance"
[13] "iter"              "weights"           "prior.weights"     "df.residual"
[17] "df.null"           "y"                 "converged"         "boundary"
[21] "model"             "call"              "formula"           "terms"
[25] "data"              "offset"            "control"           "method"
[29] "contrasts"         "xlevels"
> LBwt.glm$family

Family: binomial
Link function: logit

> LBwt.glm$df.residual
[1] 185
> LBwt.glm$coefficients
(Intercept)         AGE         LWT         FTV
 1.72065242 -0.03746067 -0.01262449 -0.05843761
```

Generalized Linear Models

- ✓ Mining the model
  - Example: Perform an ANOVA on LBwt.glm.

```
> anova(LBwt.glm)
Analysis of Deviance Table

Model: binomial, link: logit

Response: LOW

Terms added sequentially (first to last)


     Df Deviance Resid. Df Resid. Dev
NULL                   188     234.67
AGE   1   2.7600       187     231.91
LWT   1   4.7886       186     227.12
FTV   1   0.1252       185     227.00
```

  - Example: Add a term to LBwt.glm and do a before and after comparison

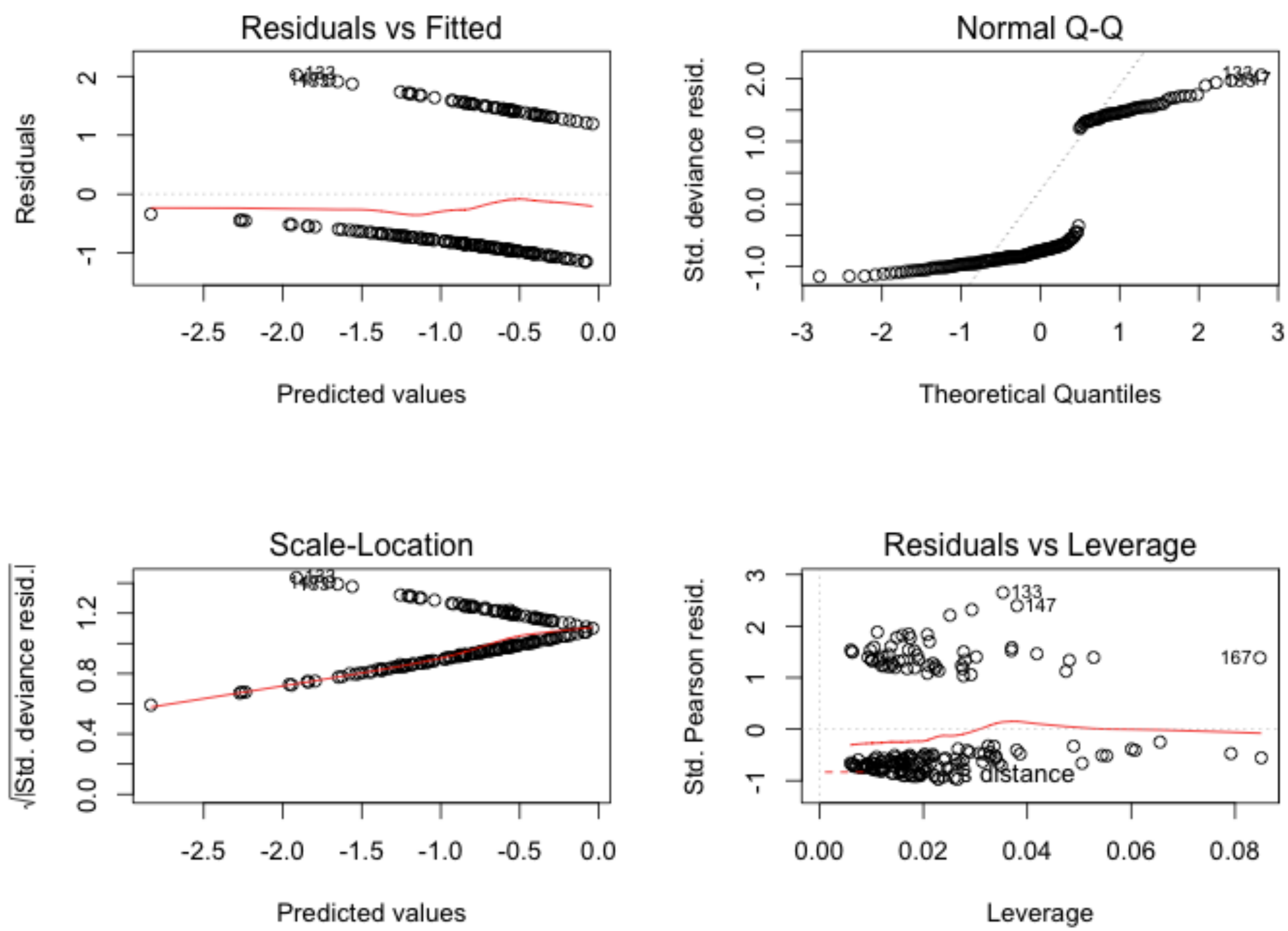
```
> LBForm
LOW ~ AGE + LWT + FTV
> LBForm1 <- LOW ~ AGE + LWT + FTV + UI
> LBwt1.glm <- glm(LBForm1,family = binomial, data=LBwt)
> anova(LBwt.glm, LBwt1.glm)
Analysis of Deviance Table

Model 1: LOW ~ AGE + LWT + FTV
Model 2: LOW ~ AGE + LWT + FTV + UI
  Resid. Df Resid. Dev Df Deviance
1       185     227.00
2       184     223.59  1   3.4046
```

Generalized Linear Models

✓ Mining the model

- Example: Perform a logistic regression on data on Low Birth Weights and the relationship to the age, weight during last menstrual period and number of physician visits during the first trimester

```
> plot(LBwt.glm)
```



Exercise

Non Linear Regression

- ✓ Introduction
  - ➡ In some cases, *glm*() can perform non-linear regression but typically, a non-linear curve fitting problems will need to be approached using non-linear optimization
  - ➡ One approach: Least squares. Lets take a case of a non linear model

$$y = \beta_1 x_1/(1 + \beta_2 x_2) + \varepsilon$$

  - ➡ In R, this is modeled as a *formula* object

```
Y <− y ~ b1*x1/(1 + b2*x2)
```

  - ➡ For *formula* objects used in non-linear models, the operators retain their usual mathematical meaning

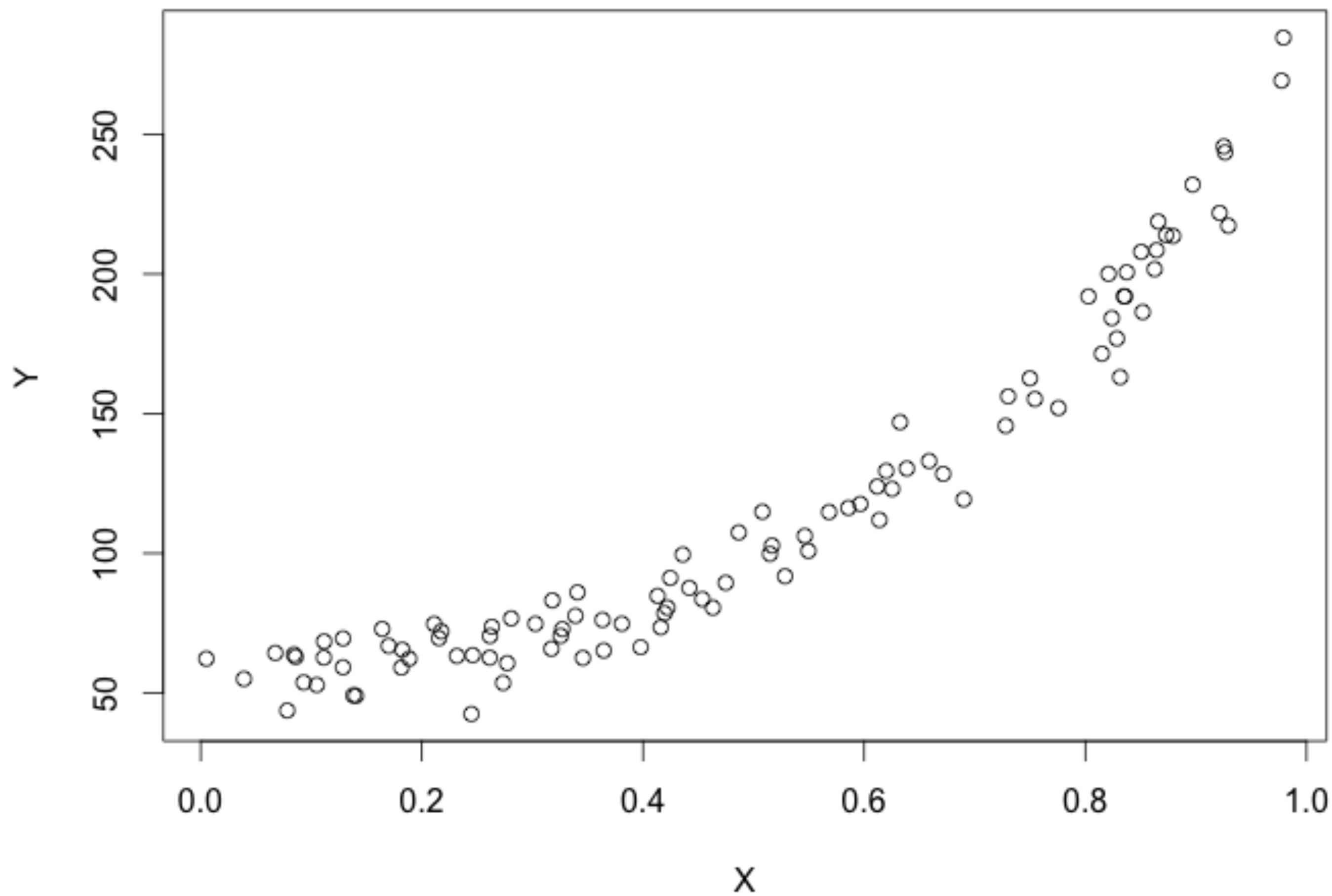
- ✓ Non Linear modeling
  - ➡ Function *nls*() may be used to generate a non-linear model
    - Usage: Type *nls*({formula}, data = , start = , weights = , subset =) from the command prompt.
      - Parameter *start* is to specify a vector of starting estimates
      - Parameter *weights* is to specify an optional vector of weights to be used in the least squares model.
      - Parameter *subset* is an optional vector specifying a subset of data to be fit.

Non Linear Regression

- ✓ Non Linear modeling

  - Example: Perform a non-linear regression on $y = e^{b1*x+b2}/(1 + b1*x)$. First step - obtain the sample data set

```
> x <- runif(100)
> y <- exp(3*x+4)/(1+3*x)
> y <- y + rnorm(100,0,10)
> nld.data <- data.frame(X = x, Y = y)
> nld.data
             X         Y
1   0.65937485 143.69201
2   0.41367685  77.80116
3   0.36812198  68.75270
4   0.10010136  56.76012
5   0.36976991  73.45288
    :
97  0.56532750 101.10755
98  0.14757301  64.84434
99  0.30649465  73.21860
100 0.29231144  75.62010
> plot(x,y)
```



Non Linear Regression

✓ Mining the model

- Example Step 2: Create the formula object and perform the regression

```
> EXPFORM <- y ~ exp(b1*x + b2)/(1 + b1*x)
> EXPFORM
y ~ exp(b1 * x + b2)/(1 + b1 * x)
> NONLIN.nls <- nls(EXPFORM, start=list(b1=2.8,b2=3.75),data=nld.data)
Warning messages:
1: In min(x) : no non-missing arguments to min; returning Inf
2: In max(x) : no non-missing arguments to max; returning -Inf
> NONLIN.nls
Nonlinear regression model
  model:  y ~ exp(b1 * x + b2)/(1 + b1 * x)
   data:  nld.data
   b1    b2
3.020 3.993
 residual sum-of-squares: 9264

Number of iterations to convergence: 4
Achieved convergence tolerance: 6.899e-07
> names(NONLIN.nls)
[1] "m"           "convInfo"    "data"        "call"        "dataClasses" "control"
```

Non Linear Regression

- Mining the model
  - Example Step 2: Obtain the summary on NONLIN.nls

```
> summary(NONLIN.nls)

Formula: y ~ exp(b1 * x + b2)/(1 + b1 * x)

Parameters:
   Estimate Std. Error t value Pr(>|t|)
b1  3.02022    0.03824   78.98   <2e-16 ***
b2  3.99281    0.02205  181.10   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 9.723 on 98 degrees of freedom

Number of iterations to convergence: 4
Achieved convergence tolerance: 6.899e-07
```

  - Example: From NONLIN.nls, obtain some attributes and their values

```
> NONLIN.nls$convInfo
$isConv
[1] TRUE

$finIter
[1] 4

$finTol
[1] 6.899151e-07

$stopCode
[1] 0

$stopMessage
[1] "converged"
```

```
> NONLIN.nls$call
nls(formula = EXPFORM, data = nld.data, start = list(b1 = 2.8,
    b2 = 3.75), algorithm = "default", control = list(maxiter = 50,
    tol = 1e-05, minFactor = 0.0009765625, printEval = FALSE,
    warnOnly = FALSE), trace = FALSE)
> NONLIN.nls$dataClasses
        x
"numeric"
```

Non Linear Regression

- ✓ Mining the model
  - ➡ Functions *coef*(), *resid*(), *fitted*() may be used to obtain the co-efficients, residuals, and fitted values respectively of a model
    - Usage: Type *coef*({model}) or *resid*({model}) or *fitted*({model}) from the command prompt.
    - Example: For NONLIN.nls

```
> coef(NONLIN.nls)
      b1       b2
3.020217 3.992805
> resid(NONLIN.nls)
  [1]  10.93769286  -6.25945546  -9.27537422   0.44406815  -4.78014355  -8.52922300  -2.12455796
  [8]  -0.35403589  21.32239913  -0.97853704  -0.59104601 -22.32897439   2.37064282  -7.12279991
 [15] -18.84117567   8.56449415   7.19097975   2.61957295   7.22335296  11.66719379   5.82967677
 [22]  -4.44157508  -5.22392417   5.69209600  -2.72553492   3.13066252  -9.10869458   5.01057017
 [29]   0.19012460  -1.90805813   2.27183417 -15.88514440  12.55471359   9.94205305  -9.41712665
 [36] -10.63187358  11.50014429   0.40763582  11.00240450  -0.76013885  -9.10974358 -21.26848525
 [43]  13.58428483   3.13112730  15.92089269   2.54989552   4.35075337 -17.26622034  -5.22821923
 [50]   6.65487552  -1.98042095  -8.76236487  -0.23765118   3.00398500  -4.05308276  -6.40066848
 [57]  -5.53600835 -10.91914620 -12.35351142 -33.57495445  -3.01427149  16.39860217  -5.27505887
 [64]  -0.06267565  -5.50689753  -2.79979214  -8.15384215   1.61981399  -0.37298104  15.88521582
 [71]   7.26861837   7.73765857   8.62850421  13.85979820 -14.25175401   8.86630572   4.59141118
 [78]  16.85633117  13.05635061  -6.46902842   0.60237973 -10.38260129  -2.31090938  11.18719506
 [85] -14.10088784 -10.79130899   5.75889652   3.52894976  13.30131174  -0.43635440  -6.09555850
 [92]   3.09123926   8.64244119   2.47979188   4.41177341  11.41303937  -9.30498277   6.29252156
 [99]   2.18123502   6.01306167
attr(,"label")
[1] "Residuals"
```

Non Linear Regression

- ✓ Mining the model
  - Example: For NONLIN.nls, the fitted values are

```
> fitted(NONLIN.nls)
  [1] 132.75432  84.06061  78.02808  56.31605  78.23302 182.15775 247.29498  57.85305 111.37047
 [10]  84.02218 207.34487  78.09419 124.76106 163.58090  60.49326 102.12035  93.91922 147.13021
 [19] 109.35731  64.61581  64.13963 207.19605  54.42036  69.25611 225.70274  55.16419  80.66665
 [28] 224.03621 215.61698 252.54782 114.42939 107.40656  58.14888 179.78156 113.41762 241.46270
 [37] 245.54694  76.12999  97.79646 131.75309 131.95420 105.90767 220.79777 125.49163  74.74456
 [46]  80.24033  73.69320 119.75293  54.97896 114.39686  68.46279 228.91997 252.81630  79.63001
 [55]  57.02792  55.75364 181.06246  96.64314  87.97207 184.73035 150.60435 148.47323 249.56976
 [64]  72.09559 151.66404 105.71622  77.55507  72.47948 124.67248 256.22779  73.10558  55.31093
 [73] 165.16942 108.54314 131.07875  87.32964  54.43915  71.12825 141.58211  63.95288 171.81883
 [82] 143.39303 208.11870 149.25836 107.86514 215.32689  71.55522 112.84613  99.07104 243.24980
 [91]  57.64020 251.04081 154.03963  98.05648  87.08931 250.45716 110.41253  58.55182  71.03737
[100]  69.60704
attr(,"label")
[1] "Fitted values"
```

Non Linear Regression

✓ Mining the model

- Example: For NONLIN.nls, obtain the variance-covariance matrix of the estimated parameters

```
> vcov(NONLIN.nls)
              b1            b2
b1  0.0014623093 -0.0008003406
b2 -0.0008003406  0.0004860789
```

- Example: For NONLIN.nls, calculate model predictions and standard errors

```
> predict(NONLIN.nls)
  [1] 132.75432  84.06061  78.02808  56.31605  78.23302 182.15775 247.29498  57.85305 111.37047
 [10]  84.02218 207.34487  78.09419 124.76106 163.58090  60.49326 102.12035  93.91922 147.13021
 [19] 109.35731  64.61581  64.13963 207.19605  54.42036  69.25611 225.70274  55.16419  80.66665
 [28] 224.03621 215.61698 252.54782 114.42939 107.40656  58.14888 179.78156 113.41762 241.46270
 [37] 245.54694  76.12999  97.79646 131.75309 131.95420 105.90767 220.79777 125.49163  74.74456
 [46]  80.24033  73.69320 119.75293  54.97896 114.39686  68.46279 228.91997 252.81630  79.63001
 [55]  57.02792  55.75364 181.06246  96.64314  87.97207 184.73035 150.60435 148.47323 249.56976
 [64]  72.09559 151.66404 105.71622  77.55507  72.47948 124.67248 256.22779  73.10558  55.31093
 [73] 165.16942 108.54314 131.07875  87.32964  54.43915  71.12825 141.58211  63.95288 171.81883
 [82] 143.39303 208.11870 149.25836 107.86514 215.32689  71.55522 112.84613  99.07104 243.24980
 [91]  57.64020 251.04081 154.03963  98.05648  87.08931 250.45716 110.41253  58.55182  71.03737
[100]  69.60704
```

Non Linear Regression

- ✓ Obtaining information
  - ➡ Other functions:

Function	Purpose
deviance	Returns the deviance of a fitted model
confint	Returns confidence intervals for model parameters
formula	Generic function for extracting formulae
step	Choose a model in a stepwise algorithm
proj	Returns a matrix of projections of data onto the model terms

Exercise

Tree Models

- ✓ Introduction
  - ➡ It may be sufficient to classify and bifurcate the data at critical points, i.e., forming a tree model, as opposed to seeking an explicit regression model
  - ➡ This approach is easier to understand, especially when there are numeric as well as categorical variables involved
  - ➡ Package "rpart" is a user-contributed package that may be used to create tree models
- ✓ Tree models
  - ➡ Function rpart() may be used to generate a tree model
    - Usage: Type rpart({formula}, data = , weights = , subset =) from the command prompt.
      - Parameter *weights* is to specify an optional vector of weights to be used in the least squares model.
      - Parameter *subset* is an optional vector specifying a subset of data to be fit.
    - Example: Using Edgar Anderson's Iris data, create a tree classification for *Species*

```
> iris.FORM <- Species ~ Sepal.Length + Sepal.Width + Petal.Length + Petal.Width
> iris.FORM
Species ~ Sepal.Length + Sepal.Width + Petal.Length + Petal.Width
> iris.TREE <- rpart(iris.FORM, data=iris)
> iris.TREE
n= 150

node), split, n, loss, yval, (yprob)
      * denotes terminal node

1) root 150 100 setosa (0.33333333 0.33333333 0.33333333)
  2) Petal.Length< 2.45 50   0 setosa (1.00000000 0.00000000 0.00000000) *
  3) Petal.Length>=2.45 100  50 versicolor (0.00000000 0.50000000 0.50000000)
    6) Petal.Width< 1.75 54   5 versicolor (0.00000000 0.90740741 0.09259259) *
    7) Petal.Width>=1.75 46   1 virginica (0.00000000 0.02173913 0.97826087) *
```

Tree Models

✓ Mining the model

- Example: Obtain the summary of iris.TREE

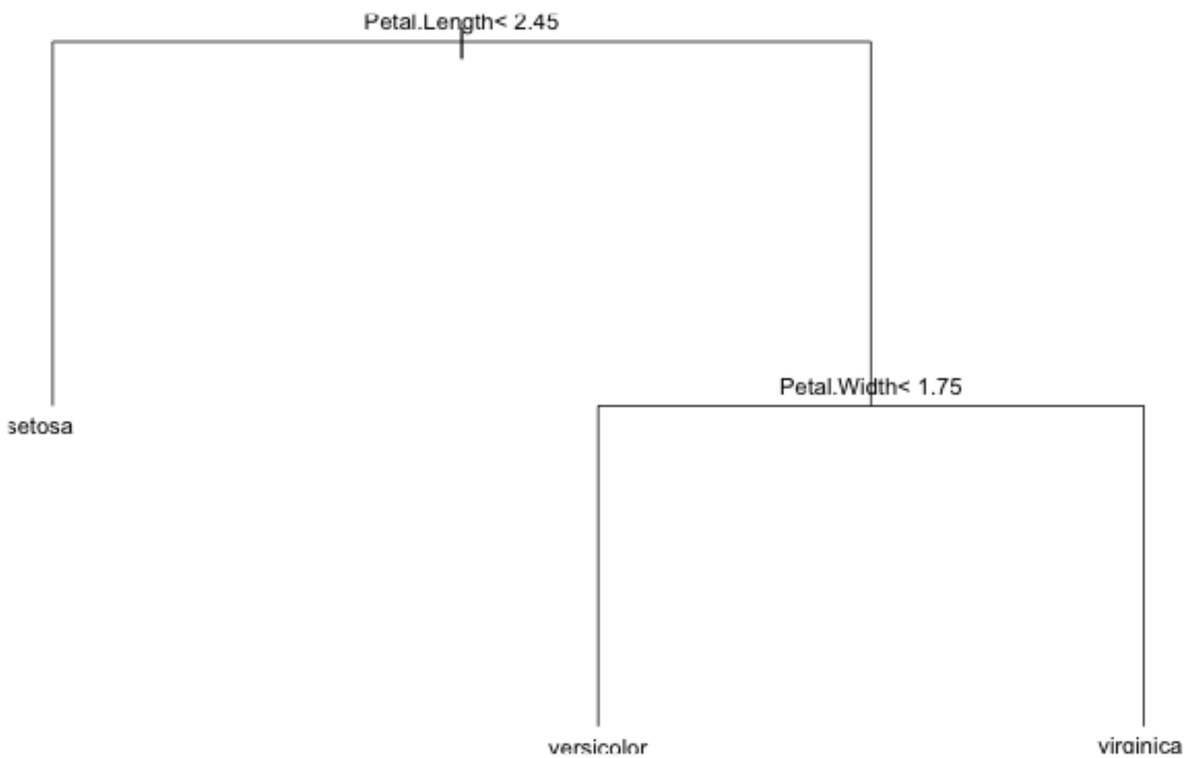
```
> summary(iris.TREE)
Call:
rpart(formula = iris.FORM, data = iris)
  n= 150

    CP nsplit rel error xerror       xstd
1 0.50      0      1.00   1.21 0.04836666
2 0.44      1      0.50   0.74 0.06123180
3 0.01      2      0.06   0.10 0.03055050

Node number 1: 150 observations,    complexity param=0.5
  predicted class=setosa      expected loss=0.6666667
    class counts:    50    50    50
   probabilities: 0.333 0.333 0.333
  left son=2 (50 obs) right son=3 (100 obs)
  Primary splits:
      Petal.Length < 2.45 to the left,  improve=50.00000, (0 missing)
      Petal.Width  < 0.8  to the left,  improve=50.00000, (0 missing)
      Sepal.Length < 5.45 to the left,  improve=34.16405, (0 missing)
      Sepal.Width  < 3.35 to the right, improve=19.03851, (0 missing)
  Surrogate splits:
      Petal.Width  < 0.8  to the left,  agree=1.000, adj=1.00, (0 split)
      Sepal.Length < 5.45 to the left,  agree=0.920, adj=0.76, (0 split)
      Sepal.Width  < 3.35 to the right, agree=0.833, adj=0.50, (0 split)
            :
Node number 6: 54 observations
  predicted class=versicolor  expected loss=0.09259259
    class counts:     0    49     5
   probabilities: 0.000 0.907 0.093

Node number 7: 46 observations
  predicted class=virginica   expected loss=0.02173913
    class counts:     0     1    45
   probabilities: 0.000 0.022 0.978
```

Tree Models

✓ Graphing the model

- Example: Obtain a plot of iris.TREE

```
> plot(iris.TREE)
> text(iris.TREE,cex=0.7)
```



Tree Models

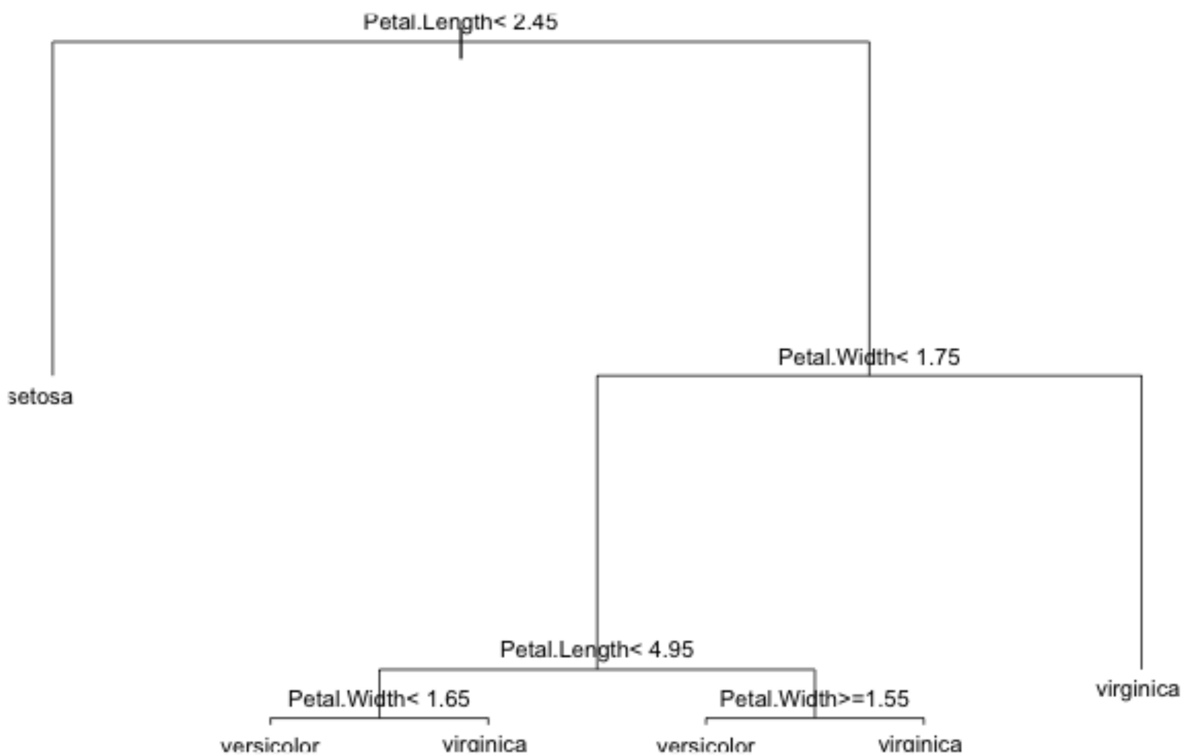
- ✓ Tweaking the tree structure
  - ➡ Function *rpart.control*() may be used to control how the algorithm evaluates nodes and splits in the tree structure
    - Usage: Used within the *rpart*() function *control* parameter
    - Type rpart({formula}, data = , control = *rpart.control*(), weights = , subset =) from the command prompt.
    - Parameters for rpart.control()
      - Parameter *minsplit* specifies the minimum number of observations that must exist in a node for a split to occur
      - Parameter *minbucket* specifies the minimum number of observations in a terminal node (leaf)
      - Parameter *cp* specifies the minimum improvement to fit for a split to be considered

Tree Models

✓ Tweaking the tree structure

- Example: In iris.TREE, change the *minsplit* to 4 from 20 and *cp* to 0.0001

```
> iris.TREE <- rpart(iris.FORM, data=iris,control=rpart.control(minsplit=4,cp=0.0001))
> plot(iris.TREE)
> text(iris.TREE,cex=0.7)
```

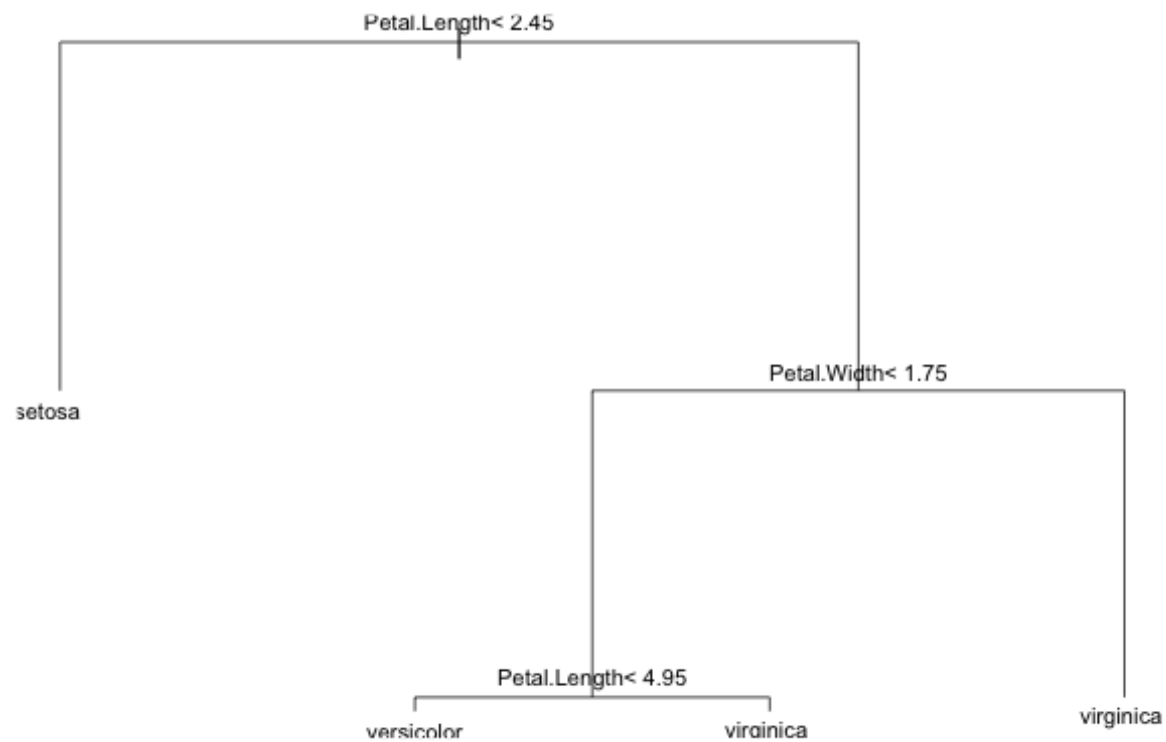


Tree Models

✓ Tweaking the tree structure

- Example: In iris.TREE, change the *minbucket* to 5 and keep the changes to *minsplit* and *cp*.

```
> iris.TREE <- rpart(iris.FORM, data=iris,control=rpart.control(minsplit=4,cp=0.0001, minbucket=5))
> plot(iris.TREE)
> text(iris.TREE,cex=0.7)
```



Exercise