同濟大學
TONGJI UNIVERSITY

生物信息学系
DEPARTMENT OF BIOINFORMATICS

Section 12

R: Probability
General Statistics

张勇

yzhang@tongji.edu.cn

同济大学

2021年5月31日

Names for probability distributions

Function	Purpose
dnorm	Normal density
pnorm	Normal distribution function
qnorm	Normal quantile function
rnorm	Normal random variates

Discrete distributions

Discrete distribution	R name	Parameters
Binomial	binom	n = number of trials; p = probability of success for one trial
Geometric	geom	p = probability of success for one trial
Hypergeometric	hyper	m = number of white balls in urn; n = number of black balls in urn; k = number of balls drawn from urn
Negative binomial (NegBinomial)	nbinom	size = number of successful trials; either prob = probability of successful trial or mu = mean
Poisson	pois	lambda = mean

Continuous distribution	R name	Parameters
Beta	beta	shape1; shape2
Cauchy	cauchy	location; scale
Chi-squared (Chisquare)	chisq	df = degrees of freedom
Exponential	exp	rate
F	f	df1 and df2 = degrees of freedom
Gamma	gamma	rate; either rate or scale
Log-normal (Lognormal)	lnorm	meanlog = mean on logarithmic scale; sdlog = standard deviation on logarithmic scale
Logistic	logis	location; scale
Normal	norm	mean; sd = standard deviation
Student's *t* (TDist)	t	df = degrees of freedom
Uniform	unif	min = lower limit; max = upper limit
Weibull	weibull	shape; scale
Wilcoxon	wilcox	m = number of observations in first sample; n = number of observations in second sample

Counting the Number of Combinations

- **Problem**
  - You want to calculate the number of combinations of *n* items taken *k* at a time.

- **Solution**
  - Use the **choose** function:

```
> choose(5,3) # How many ways select 3 items from 5 items?
[1] 10
> choose(50,3) # How many ways select 3 items from 50 items?
[1] 19600
> choose(50,30) # How many ways select 30 items from 50 items?
[1] 4.712921e+13
```

Generating Combinations

- **Problem**
  - You want to generate all combinations of *n* items taken *k* at a time.

- **Solution**
  - Use the **combn** function:

```
> combn(c("T1","T2","T3","T4","T5"), 3)
     [,1]  [,2]  [,3]  [,4]  [,5]  [,6]  [,7]  [,8]  [,9]  [,10]
[1,] "T1"  "T1"  "T1"  "T1"  "T1"  "T1"  "T2"  "T2"  "T2"  "T3"
[2,] "T2"  "T2"  "T2"  "T3"  "T3"  "T4"  "T3"  "T3"  "T4"  "T4"
[3,] "T3"  "T4"  "T5"  "T4"  "T5"  "T5"  "T4"  "T5"  "T5"  "T5"
```

Generating Random Numbers

- **Problem**
  - You want to generate random numbers.
- **Solution**
  - R can generate random variates from all distributions. For a given distribution, the name of the random number generator is “r” prefixed to the distribution’s abbreviated name:

```
> runif(1)
[1] 0.4987063
> runif(3)
[1] 0.6392485 0.4608283 0.5143987
> runif(3, min=-3, max=3)
[1]  1.996462  2.346558 -1.648518
```

Generating Random Numbers

```
> rnorm(1)
[1] -1.386023
> rnorm(1, mean=100, sd=15)
[1] 107.3787
> rbinom(1, size=10, prob=0.5)
[1] 4
> rpois(1, lambda=10)
[1] 8
> rexp(1, rate=0.1)
[1] 17.89515
> rgamma(1, shape=2, rate=0.1)
[1] 11.77107
> rnorm(3, mean=c(-10,0,+10), sd=1)
[1] -9.398185  1.865719 10.126261
```

Generating Reproducible Random Numbers

- **Problem**
  - You want to generate a sequence of random numbers, but you want to reproduce the same sequence every time your program runs.
- **Solution**
  - Before running your R code, call the **set.seed** function to initialize the random number generator to a known state:

```
> runif(3)
[1] 0.6106368 0.6159386 0.4261986
> set.seed(165)
> runif(3)
[1] 0.1159132 0.4498443 0.9955451
> set.seed(165)
> runif(3)
[1] 0.1159132 0.4498443 0.9955451
```

Generating a Random Sample

- **Problem**
  - You want to sample a dataset randomly.

- **Solution**
  - The **sample** function will randomly select *n* items from a vector without replacement:

```
> sample(1900:2000, 10)
 [1] 1967 1916 1977 1943 1990 1949 1902 1975 1966 1937
```

Generating a Random Sample

- Use the **sample** function with replace=TRUE to generate a random sequence:

```
> sample(c("H","T"), 10, replace=TRUE)
 [1] "H" "T" "T" "H" "H" "T" "H" "H" "T" "H"
> sample(c(FALSE,TRUE), 10, replace=TRUE, prob=c(0.2,0.8))
[1]  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE
```

- Use the **sample** function to generate a random permutation of a vector:

```
> sample(1:10)
 [1]  2  5  8  1  6  9 10  3  7  4
```

Calculating Probabilities for Discrete Distributions

- **Problem**
  - You want to calculate either the simple or the cumulative probability associated with a discrete random variable.

- **Solution**
  - For a simple probability, P(X = x), use the density function. All built-in probability distributions have a density function whose name is “d” prefixed to the distribution name:

```
> dbinom(7, size=10, prob=0.5)
[1] 0.1171875
```

Calculating Probabilities for Discrete Distributions

– For a cumulative probability, P(X ≤ x), use the distribution function. All built-in probability distributions have a distribution function whose name is “p” prefixed to the distribution name:

```
> pbinom(7, size=10, prob=0.5)    # P(X <= 7)
[1] 0.9453125
> pbinom(7, size=10, prob=0.5, lower.tail=FALSE)    # P(X > 7)
[1] 0.0546875
> # P(3 <X<= 7)
> pbinom(7,size=10,prob=0.5) - pbinom(3,size=10,prob=0.5)
[1] 0.7734375
> pbinom(c(3,7), size=10, prob=0.5)
[1] 0.1718750 0.9453125
> diff(pbinom(c(3,7), size=10, prob=0.5))
[1] 0.7734375
```

Calculating Probabilities for Continuous Distributions

- **Problem**
  - You want to calculate the distribution function (DF) or cumulative distribution function (CDF) for a continuous random variable.

- **Solution**
  - Use the distribution function, which calculates $P(X \leq x)$. All built-in probability distributions have a distribution function whose name is “p” prefixed to the distribution’s abbreviated name:

```
> pnorm(66, mean=70, sd=3)     # P(X <= 66), given X ~ N(70, 3)
[1] 0.09121122
```

Converting Probabilities to Quantiles

- **Problem**
  - Given a probability p and a distribution, you want to determine the corresponding quantile for p: the value x such that $P(X \leq x) = p$.

- **Solution**
  - Every built-in distribution includes a quantile function that converts probabilities to quantiles. The function's name is "q" prefixed to the distribution name:

```
> qnorm(0.05, mean=100, sd=15)
[1] 75.3272
> qnorm(c(0.025,0.975))
[1] -1.959964  1.959964
```

Plotting a Density Function

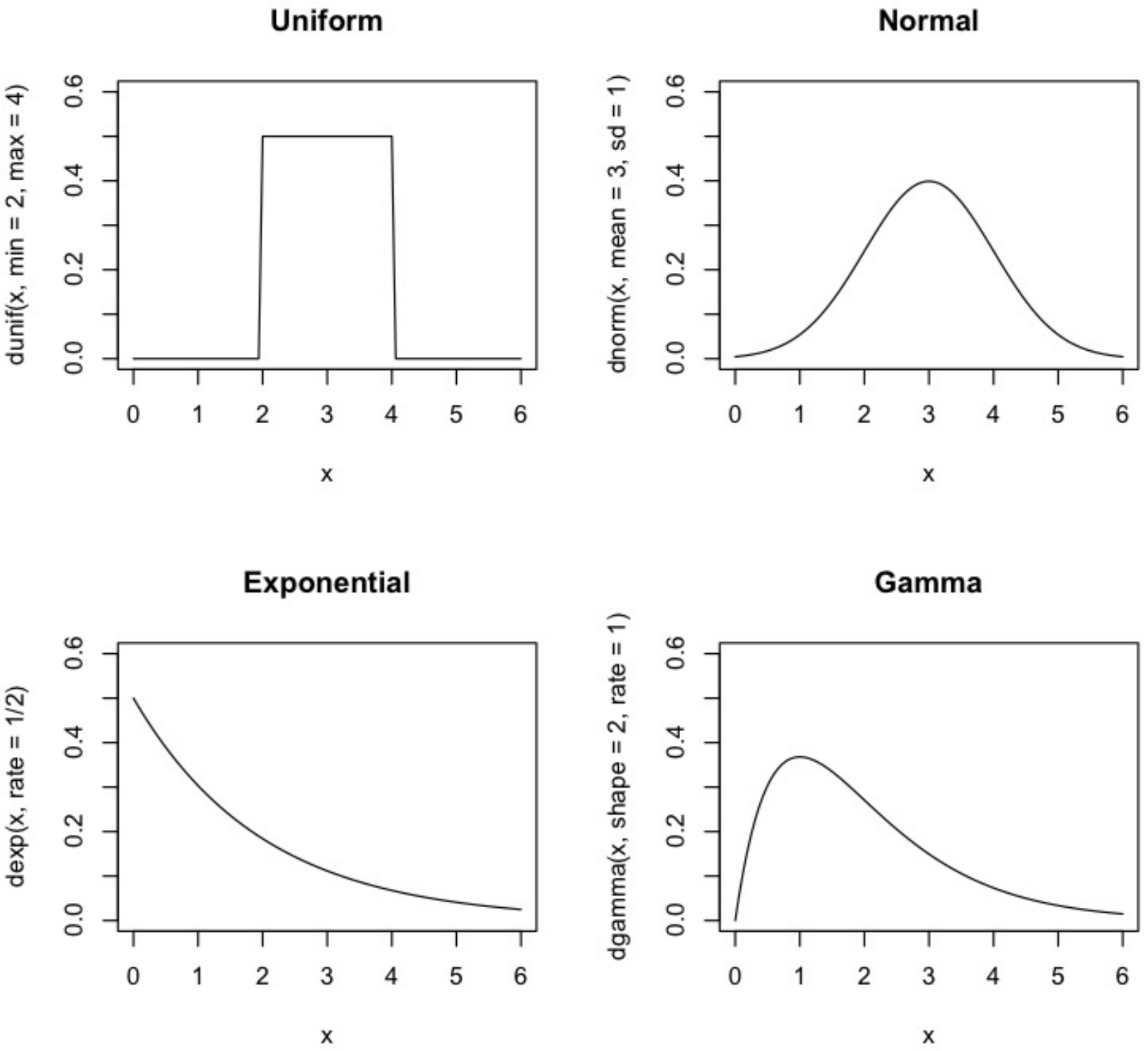
- **Problem**
  - You want to plot the density function of a probability distribution.

- **Solution**
  - Define a vector x over the domain. Apply the distribution's density function to x and then plot the result:

```
> x <- seq(from=-3, to=3, length.out=100)
> plot(x, dnorm(x))
```

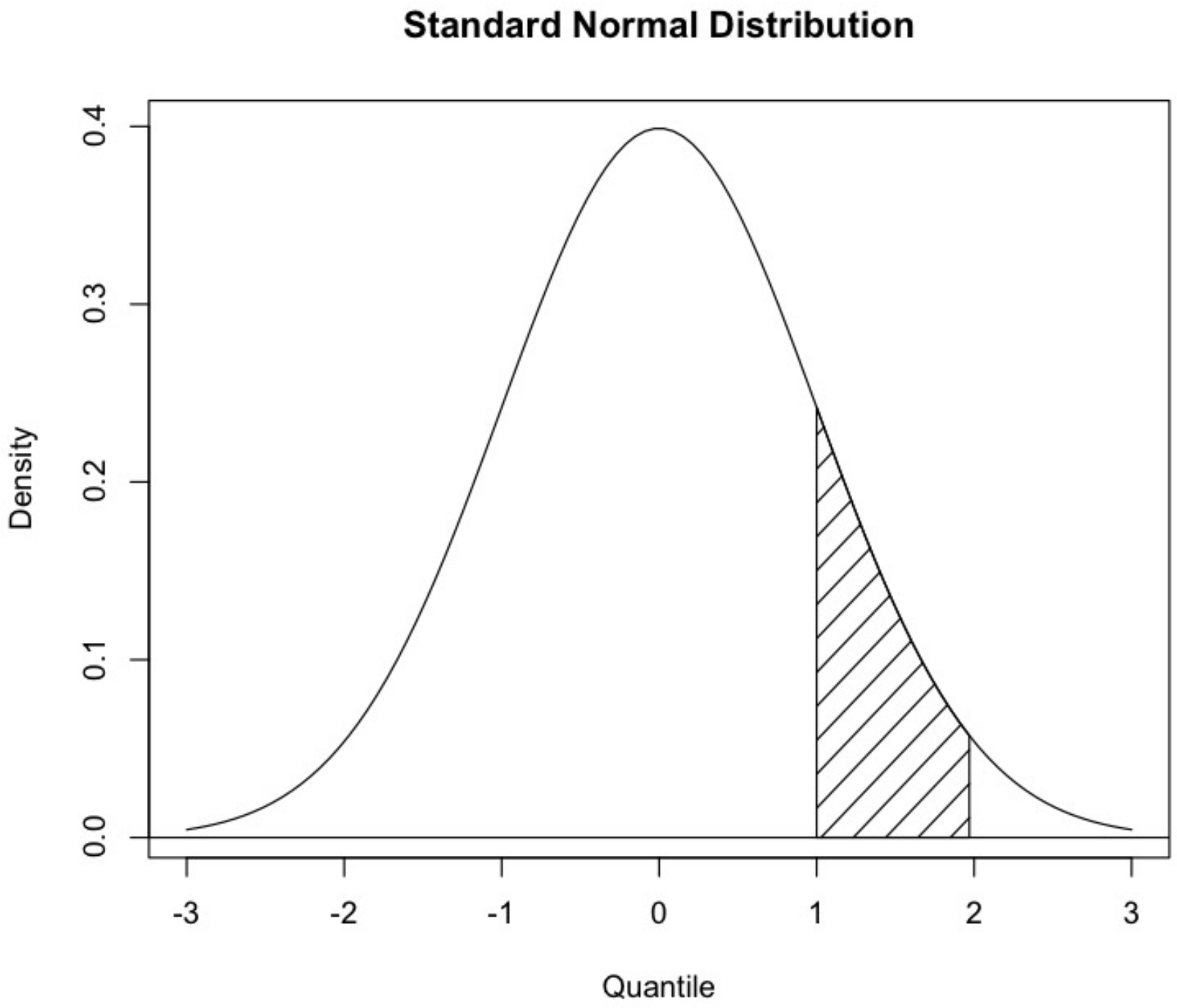
Plotting a Density Function

```
> x <- seq(from=0, to=6, length.out=100)
> ylim <- c(0, 0.6)
> par(mfrow=c(2,2))
> plot(x, dunif(x,min=2,max=4), main="Uniform", type='l', ylim=ylim)
> plot(x, dnorm(x,mean=3,sd=1), main="Normal", type='l', ylim=ylim)
> plot(x, dexp(x,rate=1/2), main="Exponential", type='l', ylim=ylim)
> plot(x, dgamma(x,shape=2,rate=1), main="Gamma", type='l', ylim=ylim)
```


Uniform
dunif(x, min = 2, max = 4)
0.0
0.2
0.4
0.6
0
1
2
3
4
5
6
x
Normal
dnorm(x, mean = 3, sd = 1)
0.0
0.2
0.4
0.6
0
1
2
3
4
5
6
x
Exponential
dexp(x, rate = 1/2)
0.0
0.2
0.4
0.6
0
1
2
3
4
5
6
x
Gamma
dgamma(x, shape = 2, rate = 1)
0.0
0.2
0.4
0.6
0
1
2
3
4
5
6
x

Plotting a Density Function

```
> x <- seq(from=-3, to=+3, length.out=100)
> y <- dnorm(x)
> plot(x, y, main="Standard Normal Distribution", type='l',
+ ylab="Density", xlab="Quantile")
> abline(h=0)
> # The body of the polygon follows the density curve where 1 <= z <= 2
> region.x <- x[1 <= x & x <= 2]
> region.y <- y[1 <= x & x <= 2]
> # We add initial and final segments, which drop down to the Y axis
> region.x <- c(region.x[1], region.x, tail(region.x,1))
> region.y <- c( 0, region.y, 0)
> polygon(region.x, region.y, density=10)
```


Standard Normal Distribution
Density
0.0
0.1
0.2
0.3
0.4
-3
-2
-1
0
1
2
3
Quantile

Summarizing Your Data

- **Problem**
  - You want a basic statistical summary of your data.

- **Solution**
  - The **summary** function gives some useful statistics for vectors, matrices, factors, and data frames:

```
> summary(cars)
     speed           dist       
 Min.   : 4.0   Min.   :  2.00  
 1st Qu.:12.0   1st Qu.: 26.00  
 Median :15.0   Median : 36.00  
 Mean   :15.4   Mean   : 42.98  
 3rd Qu.:19.0   3rd Qu.: 56.00  
 Max.   :25.0   Max.   :120.00
```

Calculating Relative Frequencies

- **Problem**
  - You want to count the relative frequency of certain observations in your sample.

- **Solution**
  - Identify the interesting observations by using a logical expression; then use the **mean** function to calculate the fraction of observations it identifies:

```
> x <- 1:100
> mean(x > 40)
[1] 0.6
> mean(abs(x-mean(x)) > sd(x))
[1] 0.42
```

Testing Categorical Variables for Independence

- **Problem**
  - You have two categorical variables that are represented by factors. You want to test them for independence using the chi-squared test.

- **Solution**
  - Use the **table** function to produce a contingency table from the two factors. Then use the **summary** function to perform a chi-squared test of the contingency table:

```
> summary(table(fac1, fac2))
```

Testing Categorical Variables for Independence

```
> initial <- factor(sample(c("Yes", "No", "Maybe"), 100, replace=TRUE))
> outcome <- factor(sample(c("Fail", "Pass"), 100, replace=TRUE))
> table(initial,outcome)
       outcome
initial Fail Pass
  Maybe   15   15
  No      23   16
  Yes     17   14
> summary(table(initial,outcome))
Number of cases in table: 100
Number of factors: 2
Test for independence of all factors:
      Chisq = 0.5523, df = 2, p-value = 0.7587
```

Calculating Quantiles of a Dataset

- **Problem**
  - Given a fraction *f*, you want to know the corresponding quantile of your data.
- **Solution**
  - Use the **quantile** function. The second argument is the fraction, *f*:

```
> vec <- rnorm(100)
> quantile(vec, .05)
       5%
-1.427059
> quantile(vec, c(.05, .95))
       5%       95%
-1.427059  1.597947
> quantile(vec)
          0%          25%          50%          75%         100%
-2.464185815 -0.593952009  0.007036115  0.734806991  2.219629234
```

Inverting a Quantile

- **Problem**
  - Given an observation x from your data, you want to know its corresponding quantile. That is, you want to know what fraction of the data is less than x.
- **Solution**
  - Use **mean** to compute the relative frequency of values less than x:

    ```
    > mean(vec < x)
    ```

  - The expression vec < x compares every element of vec against x and returns a vector of logical values. The **mean** function converts those logical values to 0 and 1: 0 for FALSE and 1 for TRUE. The average of all those 1s and 0s is the fraction of vec that is less than x, or the inverse quantile of x.

Converting Data to Z-Scores

- **Problem**
  - You have a dataset, and you want to calculate the corresponding z-scores for all data elements. (This is sometimes called *normalizing the data*.)

- **Solution**
  - Use the **scale** function for vectors, matrices, and data frames. In the case of matrices and data frames, **scale** normalizes each column independently and returns columns of normalized values in a matrix:

```
> scale(x)
```

Testing the Mean of a Sample

- **Problem**
  - You have a sample from a population. Given this sample, you want to know if the mean of the population could reasonably be a particular value *m*.

- **Solution**
  - Apply the **t.test** function to the sample *x* with the argument mu=*m*:

```
> t.test(x, mu=m)
```

Testing the Mean of a Sample

```
> x <- rnorm(50, mean=100, sd=15)
> t.test(x, mu=95)

        One Sample t-test

data:  x
t = 2.701, df = 49, p-value = 0.009466
alternative hypothesis: true mean is not equal to 95
95 percent confidence interval:
  96.1025 102.5111
sample estimates:
mean of x
  99.3068
```

Testing a Sample Proportion

- **Problem**
  - You have a sample of values from a population consisting of successes and failures. You believe the true proportion of successes is *p*, and you want to test that hypothesis using the sample data.

- **Solution**
  - Use the **prop.test** function. Suppose the sample size is *n* and the sample contains *x* successes:

```
> prop.test(x, n, p)
```

Testing a Sample Proportion

```
> prop.test(11, 20, 0.5, alternative="greater")

        1-sample proportions test with continuity correction

data:  11 out of 20, null probability 0.5
X-squared = 0.05, df = 1, p-value = 0.4115
alternative hypothesis: true p is greater than 0.5
95 percent confidence interval:
 0.3496150 1.0000000
sample estimates:
   p
0.55
```

Testing for Normality

- **Problem**
  - You want a statistical test to determine whether your data sample is from a normally distributed population.

- **Solution**
  - Use the **shapiro.test** function:

```
> shapiro.test(x)

        Shapiro-Wilk normality test

data:  x
W = 0.9873, p-value = 0.863
```

Testing for Runs

- **Problem**
  - Your data is a sequence of binary values: yes-no, 0-1, true-false, or other two-valued data. You want to know: is the sequence random?

- **Solution**
  - The **tseries** package contains the **runs.test** function, which checks a sequence for randomness. The sequence should be a factor with two levels:

```
> library(tseries)
> runs.test(as.factor(s))
```

Testing for Runs

```
> library(tseries)
> s <- sample(c(0,1), 100, replace=T)
> runs.test(as.factor(s))

        Runs Test

data:  as.factor(s)
Standard Normal = 1.0451, p-value = 0.296
alternative hypothesis: two.sided

> s <- c(0,0,0,0,0,1,1,1,1,1,0,0,0,0)
> runs.test(as.factor(s))

        Runs Test

data:  as.factor(s)
Standard Normal = -2.7029, p-value = 0.006873
alternative hypothesis: two.sided
```

Comparing the Means of Two Samples

- **Problem**
  - You have one sample each from two populations. You want to know if the two populations could have the same mean.

- **Solution**
  - Perform a *t* test by calling the **t.test** function:

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

  - By default, t.test assumes that your data are not paired. If the observations are paired, then specify paired=TRUE:

    ```
    > t.test(x, y, paired=TRUE)
    ```

Comparing the Means of Two Samples

```
> x <- rnorm(50, mean=100, sd=15)
> y <- x-rnorm(50, mean=2)
> t.test(x, y)

        Welch Two Sample t-test

data:  x and y
t = 0.7273, df = 97.999, p-value = 0.4688
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -3.797224  8.190805
sample estimates:
mean of x mean of y
101.84727  99.65048
```

Comparing the Means of Two Samples

```
> t.test(x, y, paired=TRUE)

        Paired t-test

data:  x and y
t = 16.7804, df = 49, p-value < 2.2e-16
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 1.933709 2.459871
sample estimates:
mean of the differences
                2.19679
```

Comparing the Locations of Two Samples Nonparametrically

- **Problem**
  - You have samples from two populations. You don't know the distribution of the populations, but you know they have similar shapes. You want to know: is one population shifted to the left or right compared with the other?

- **Solution**
  - You can use a nonparametric test, the Wilcoxon-Mann-Whitney test, which is implemented by the **wilcox.test** function. For paired observations, set paired=TRUE:

    ```
    > wilcox.test(x, y, paired=TRUE)
    ```

  - For unpaired observations:

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

Comparing the Locations of Two Samples Nonparametrically

```
> wilcox.test(x, y, paired=TRUE)

        Wilcoxon signed rank test with continuity correction

data:  x and y
V = 1272, p-value = 9.347e-10
alternative hypothesis: true location shift is not equal to 0

> wilcox.test(x, y)

        Wilcoxon rank sum test with continuity correction

data:  x and y
W = 1370, p-value = 0.4100
alternative hypothesis: true location shift is not equal to 0
```

Testing a Correlation for Significance

- **Problem**
  - You calculated the correlation between two variables, but you don't know if the correlation is statistically significant.

- **Solution**
  - The **cor.test** function can calculate both the p-value and the confidence interval of the correlation. If the variables came from normally distributed populations then use the default measure of correlation, i.e. Pearson method:

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

  - For nonnormal populations, use the Spearman method:

    ```
    > cor.test(x, y, method="spearman"):
    ```

Testing a Correlation for Significance

```
> cor.test(Hirsch$hr0_D2, Hirsch$hr0_D3)

        Pearson's product-moment correlation

data:  Hirsch$hr0_D2 and Hirsch$hr0_D3
t = 915.0732, df = 11522, p-value < 2.2e-16
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.9929378 0.9934336
sample estimates:
      cor
0.9931902
```

Testing Groups for Equal Proportions

- **Problem**
  - You have samples from two or more groups. The group's elements are binary-valued: either success or failure. You want to know if the groups have equal proportions of successes.
- **Solution**
  - Use the **prop.test** function with two vector arguments:

    ns <- c(ns1, ns2, ..., nsN)

    nt <- c(nt1, nt2, ..., ntN)

    prop.test(ns, nt)
  - These are parallel vectors. The first vector, **ns**, gives the number of successes in each group. The second vector, **nt**, gives the size of the corresponding group (often called the number of trials)

Testing Groups for Equal Proportions

```
> Score_85 <- c(4, 5, 1, 1, 1, 7, 16, 2)
> Total_number <- c(21, 31, 18, 14, 14, 16, 36, 29)
> prop.test(Score_85, Total_number)
```

8-sample test for equality of proportions without continuity correction

data: Score_85 out of Total_number
X-squared = 27.018, df =7, p-value = 0.0003309
alternative hypothesis: two.sided
sample estimates:

prop 1	prop 2	prop 3	prop 4	prop 5	prop 6	prop 7	prop 8
0.19047619	0.16129032	0.05555556	0.07142857	0.07142857	0.43750000	0.44444444	0.06896552

Testing Two Samples for the Same Distribution

- **Problem**
  - You have two samples, and you are wondering: did they come from the same distribution?

- **Solution**
  - The Kolmogorov-Smirnov test compares two samples and tests them for being drawn from the same distribution. The **ks.test** function implements that test:

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

Summary

- R Cookbook
  - Chapter 8. Probability
  - Chapter 9. General Statistics