TOPICS(Probability Distributions & Basic Statistics)

- Normal Distribution
- Binomial Distribution
- Poisson Distribution
- Other Distributions
- Summary Statistics
- Correlation and covariance
- T-tests
- ANOVA

- Being a statistical programming language, R easily handles all the basic necessities of statistics, including drawing random numbers and calculating distribution values, means, variances, maxima and minima, correlation and t-tests.
- 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.

Normal Distribution

- The most famous, and most used, statistical distribution is the normal distribution, sometimes referred to as the Gaussian distribution, which is defined as

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\left(\frac{1}{2}\right)\left[\frac{x-\mu}{\sigma}\right]^2}$$

where

μ = mean of x

σ = standard deviation of x

$\pi = 3.14159\ldots$

$e = 2.71828\ldots$

- This is the famous bell curve that describes so many phenomena in life.
- To draw random numbers from the normal distribution use the rnorm function, which optionally allows the specification of the mean and standard deviation.

```
>#10 draws from the standard 0-1 normal distribution
>rnorm(n=10)
[1]0.3746584 0.7368645 0.2408023 -0.1220292 0.6525665
[6]0.3313728 0.5401996 1.6598050 -0.7777772 0.4904597
>#10 draws from the 100-20 distribution
>rnorm(n=10, mean=100, sd=20)
[1] 94.99245 125.31772 120.70047 118.07148 111.88081 99.32752
[7] 92.36758 87.94429  115.18968 91.88554
```

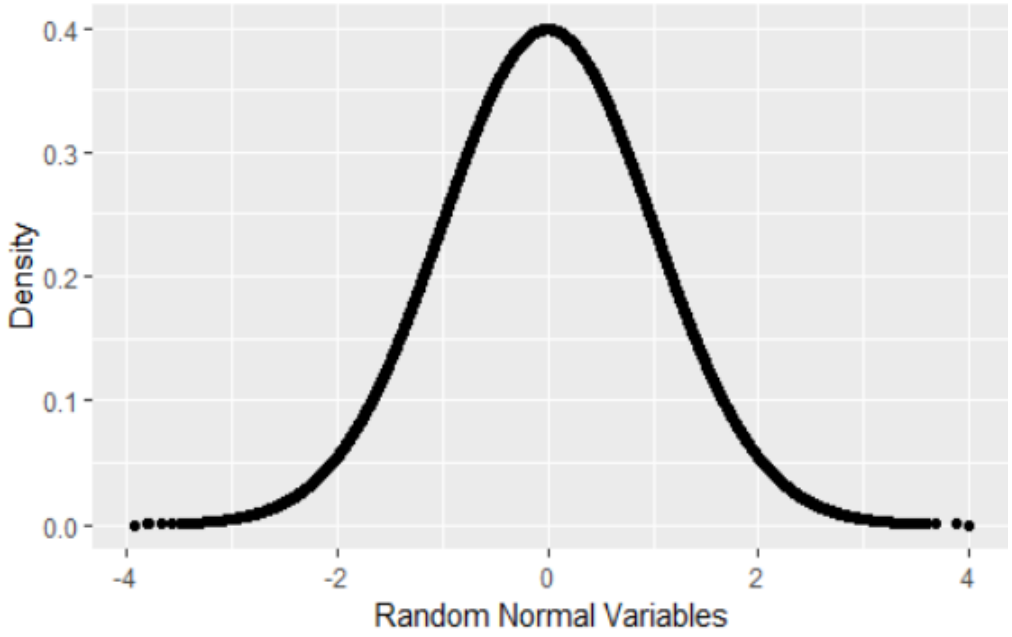
- The density for the normal distribution is calculated using dnorm.

```
>randNorm10<-rnorm(10)
randNorm10
[1] 1.8081780 0.7159731 0.4119520  -0.1659213 -0.1597631
[6] 1.0941883 0.1981299 -1.3998152 -2.2787374 -0.3403679
>dnorm(randNorm10)
[1] 0.07779389 0.30874263 0.36648754 0.39348848 0.39388328
[6] 0.21924564 0.39118830 0.14976620 0.02974005 0.37649004
>dnorm(c(-1,0,1))
[1] 0.2419707 0.3989423 0.2419707
```

- dnorm returns the probability of a specific number occurring. While it is technically mathematically impossible to find the exact probability of a number from a continuous distribution, this is an estimate of the probability. Like with rnorm, a mean and standard deviation can be specified for dnorm.

- To see this visually we generate a number of normal random variables, calculate their distributions and then plot them. This should result in a nicely shaped bell curve.

```
>#generate the normal variables
>randNorm<rnorm(30000)
>#calculate their distributions
>randDensity<-dnorm(randNorm)
>#load ggplot2
>require(ggplot2)
>#plot them
>ggplot(data.frame(x=randNorm, y=randDensity)) + aes(x=x,y=y) + geom_point() + labs(x="Random Normal Variables", y= "Density")
```



• Similarly, pnorm calculates the distribution of the normal distribution i.e, the cumulative probability that a given number, or smaller number, occurs. This is defined as

$$\Phi(a) = P\{X <= a\} = \int_{-\infty}^{a} \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-(x-\mu)^2}{2\sigma^2}} dx$$

>pnorm(randNorm10)

[1]0.96471060 0.76299601 0.65981269 0.43410943 0.43653383 0.86306380

[7]0.57852828 0.08078433 0.01134134 0.36678975

>pnorm(c(-3,0,3))

[1] 0.001349898 0.500000000 0.998650102

>pnorm(-1)

[1] 0.1586553

- By default this is left-tailed. To find the probability that the variable falls between two points, we must calculate the two probabilities and subtract them from each other.

>pnorm(1) – pnorm(0)

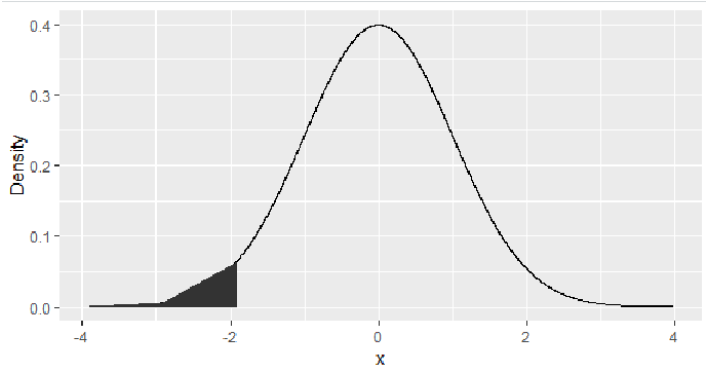
[1] 0.3413447

>pnorm(1)-pnorm(-1)

[1] 0.6826895

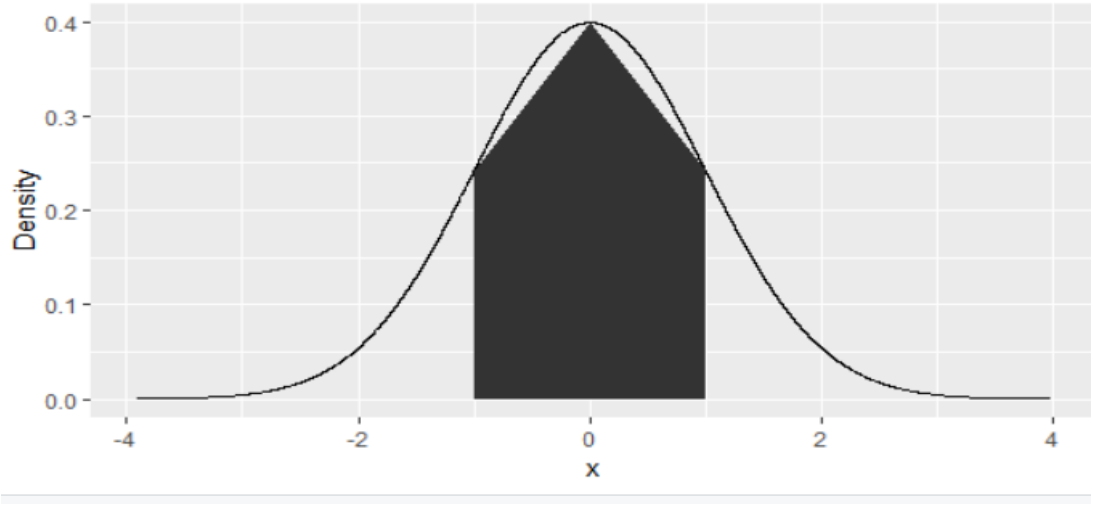
```
>#a few things happen with this first line of code
>#the idea is to build a ggplot2 object that we can build upon later
>#that is why it is saved to p
>#we take randNorm and randDensity and put them into a data.frame
>#we declare the x and y axes outside of any other function
>#this just gives more flexibility
>#we add lines with geom_line()
>#x- and y- axis labels with labs(x="x",y="Density")
>p<- ggplot(data.frame(x=randNorm,y=randDensity)) + aes(x=x,y=y)+
    geom_line()+labs(x="x",y="Density")
>#plotting p will print a nice distribution
>#to create a shaded area under the curve we first calculate that area
>#generate a sequence of numbers going from far left to -1
>neg1Seq<-seq(from=min(randNorm), to=-1, by=1)
>#build a data.frame of that sequence as x
>#the distribution values for that sequence as y
>lessThanNeg1<-data.frame(x=neg1Seq,y=dnorm(neg1Seq1))
```

```
>head(lessThanNeg1)
          x           y
1 -3.910628 0.0001905789
2 -2.910628 0.0057715394
3 -1.910628 0.0643004683
>#combine this with endpoints at the far left and far right
>#the height is 0
>lessThanNeg1<- rbind(c(min(randNorm),0),lessThanNeg1,c(max(lessThanNeg1$x),0))
>#use that shaded region as a polygon
>p+ geom_polygon(data=lessThanNeg1, aes(x=x,y=y))
```

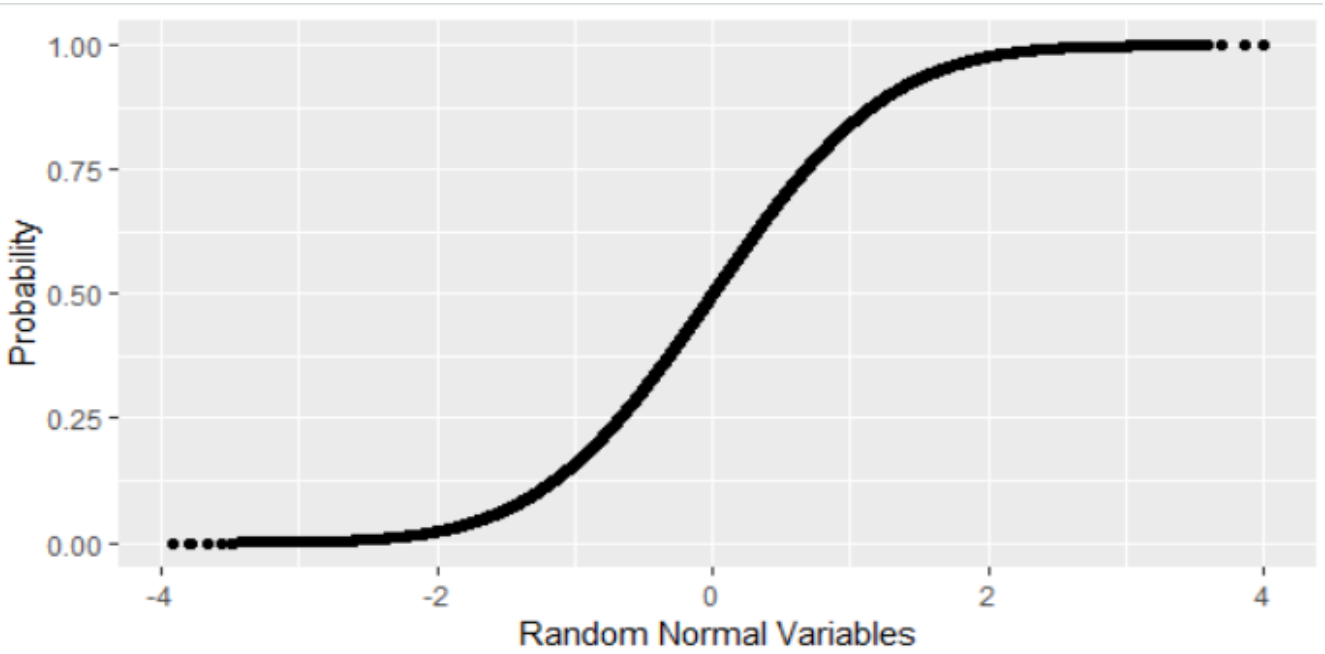



```
>#create a similar sequence going from -1 to 1
>neg1Pos1Seq<-seq(from=-1, to=1, by=1)
>#build a data.frame of that sequence as x
>#the distribution values for that sequence as y
>neg1To1<-data.frame(x=neg1Pos1Seq,y=dnorm(neg1Pos1Seq))
```

```
>head(neg1To1)
      x         y
1    -1 0.2419707
2     0 0.3989423
3     1 0.2419707
>#combine this with endpoints at the far left and far right
>#the height is 0
>neg1To1<-rbind(c(min(neg1To1$x),0),neg1To1,c(max(neg1To1$x),0))
>#use that shaded region as a polygon
>p+geom_polygon(data=neg1To1,aes(x=x,y=y))
```



```
>randProb<-pnorm(randNorm)
>ggplot(data.frame(x=randNorm,y=randProb))+aes(x=x,y=y)+ge
    om_point()+labs(x="Random Normal Variables",
    y="Probability")
```



The opposite of pnorm is qnorm. Given a cummulative probability it returns the quantile.

```
>randNorm10
[1] 1.8081780 0.7159731  0.4119520 -0.1659213 -0.1597631
[6] 1.0941883 0.1981299 -1.3998152 -2.2787374 -0.3403679
```

```
>qnorm(pnorm(randNorm10))
[1] 1.8081780 0.7159731  0.4119520 -0.1659213 -0.1597631
[6] 1.0941883 0.1981299 -1.3998152 -2.2787374 -0.3403679
>all.equal(randNorm10, qnorm(pnorm(randNorm10)))
[1] TRUE
```

Binomial Distribution

- Like the normal distribution, the binomial distribution is well represented in R. Its probability mass function is

$$p(x; n, p) = \binom{n}{x} p^x (1-p)^{n-x}$$

$$\binom{n}{x} = \frac{n!}{x!(n-x)!}$$

and n is the number of trails and p is the probability of success of a trail.

- The mean is np and the variance is np(1-p). When n=1 this reduces to the bernoulli distribution.
- Generating random numbers from the binomial distribution is not simply generating random numbers but rather generating the number of successes of independent trails.
- To simulate the number of successes out of ten trails with probability 0.4 of success, we run rbinom with n=1 (only one run of the trails), size=10 (trail size of 10), and prob=0.4 (probability of success is 0.4).

```
>rbinom(n=1,size=10,prob=0.4)
[1]  6
```

- That is to say that ten trails were conducted, each with 0.4 probability of success, and the number generated is the number that succeeded. As this is random, different numbers will be generated each time.

- By setting n to anything greater than 1, R will generate the number of successes for each of the n sets of size trails.

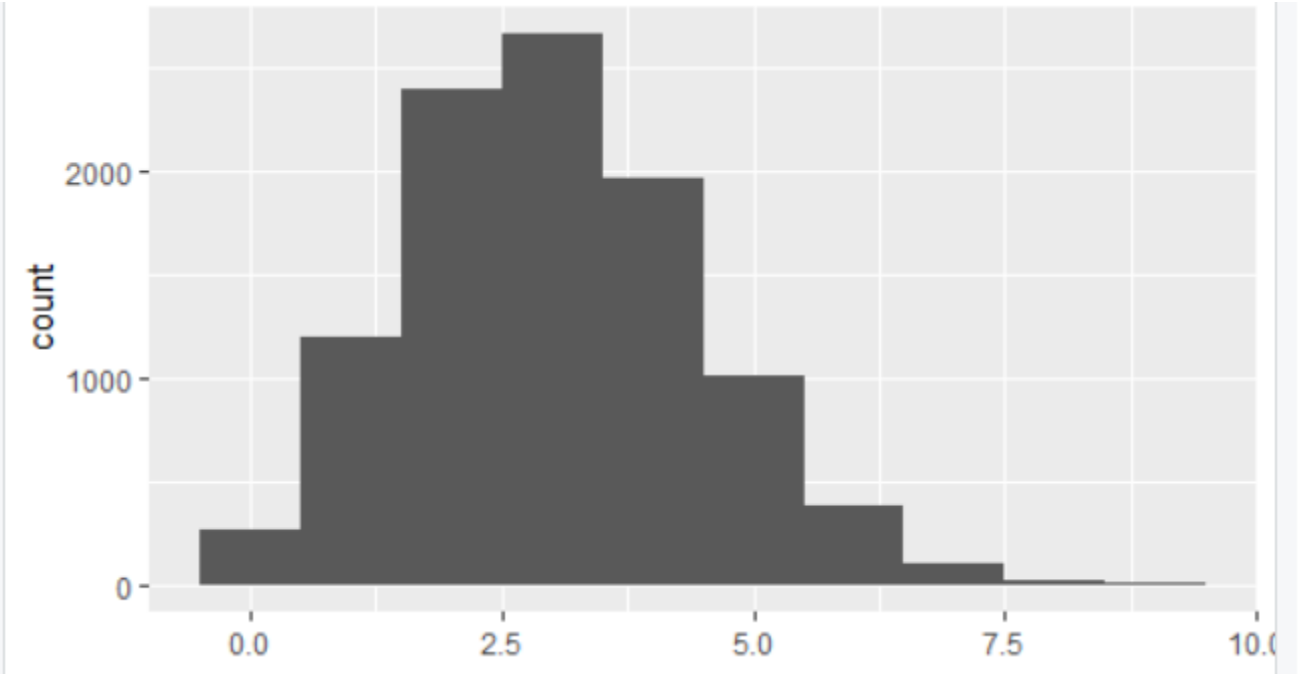
```
>rbinom(n=1,size=10,prob=0.4)
[1]  3
>rbinom(n=5,size=10,prob=0.4)
[1]  5  3  6  5  4
>rbinom(n=10,size=10,prob=0.4)
[1]  5  3  4  4  5  3  3  5  3  3
```

- Setting size to 1 turns the numbers into a Bernoulli random variable, which can take on only the value 1(success) or 0(failure).

```
>rbinom(n=1,size=1,prob=0.4)
[1]  1
>rbinom(n=5,size=1,prob=0.4)
[1]  0  0  1  1  1
>rbinom(n=10,size=1,prob=0.4)
[1]  0  0  0  1  0  1  0  0  1  0
```

- To visualize the binomial distribution we randomly generate 10,000 experiments, each with 10 trails and 0.3 probability of success.

```
>binomData<-
  data.frame(Successes=rbinom(n=10000,size=10,prob=0.3))
>ggplot(binomData, aes(x=Successes)) +
  geom_histogram(binwidth=1)
```



- To see how the binomial distribution is well approximated by the normal distribution as the number of trails grows large, we run similar experiments with differing number of trails.

```
>#create a data.frame with Successes being the 10,000 random draws
>
># Size equals 5 for all 10,000 rows
>
>binom5<-data.frame(Successes=rbinom(n=10000,
    size=5,prob=0.3),Size=5)
>
>dim(binom5)
[1] 10000   2
>
>head(binom5)
    Successes     Size
1    1                5
2    1                5
3    2                5
4    2                5
5    3                5
6    0                5
```

```
>#similar to before, still 10,000 rows
>#numbers are drawn from a distribution with a different size
>#Size now equals 10 for all 10,000 rows
>binom10<-data.frame(Successes=rbinom(n=10000,size=10,prob=0.3),
    Size=10)
>dim(binom10)
[1]  10000        2
>head(binom10)
     Successes     Size
1    1             10
2    3             10
3    3             10
4    3             10
5    0             10
6    3             10
```

```
>binom100<-
    data.frame(Successes=rbinom(n=10000,size=100,prob=0.3),Size=100)
>binom1000<-
    data.frame(Successes=rbinom(n=10000,size=100,prob=0.3),Size=1000)
>#combine them all into one data.frame
>binomAll<-rbind(binom5,binom10,binom100,binom1000)
>dim(binomAll)
[1]  40000    2
>head(binomAll, 10)
    Successes       Size
1     1                 5
2     1                 5
3     2                 5
4     2                 5
5     3                 5
6     0                 5
7     1                 5
8     1                 5
9     1                 5
10    1                 5
```

```
>tail(binomAll,10)
```

	Successes	Size
39991	316	1000
39992	311	1000
39993	296	1000
39994	316	1000
39995	288	1000
39996	286	1000
39997	264	1000
39998	291	1000
39999	300	1000
40000	302	1000

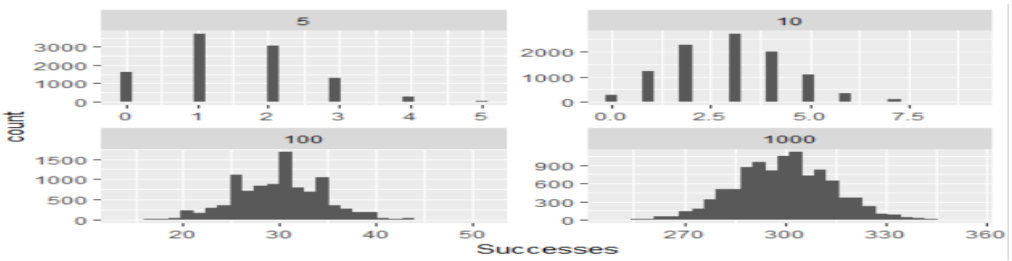
>#build the plot

>#histograms only need an x aesthetic

>#it is faceted (broken up) based on the values of Size

>#these are 5, 10, 100, 1000

>ggplot(binomAll, aes(x=Successes)) + geom_histogram()+ facet_wrap(~Size, scales="free")



- To calculate distribution function is

$$F(a; n, p) = P\{X <= a\} = \sum_{i=0}^{a} \binom{n}{i} p^i (1-p)^{n-i}$$

- Where n and p are the number of trails and the probability of success , respectively as before.
- Similar to the normal distribution functions, dbinom and pbinom provide the density (probability of an exact value) and distribution (cumulative probability), respectively, for the binomial distribution.

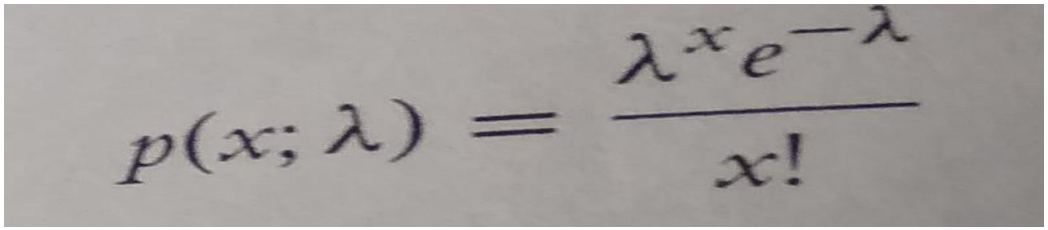
```
>#probability of 3 successes out of 10
>dbinom(x=3, size=10, prob=0.3)
[1]  0.2668279
>#probability of 3 or fewer successes out of 10
>pbinom(q=3, size=10, prob=0.3)
[1]  0.6496107
>#both functions can be vectorized
>dbinom(x=1:10, size=10, prob=0.3)
[1]  0.1210608210  0.2334744405  0.2668279320  0.2001209490  0.1029193452
[6]  0.0367569090  0.0090016920  0.0014467005  0.0001377810  0.0000059049
>pbinom(q=1:10, size=10, prob=0.3)
[1]  0.1493083  0.3827828  0.6496107  0.8497317  0.9526510  0.9894079
[7]  0.9984096  0.9998563  0.9999941  1.0000000
```

- Given a certain probability, qbinom returns the quantile, which for this distribution is the number of successes.

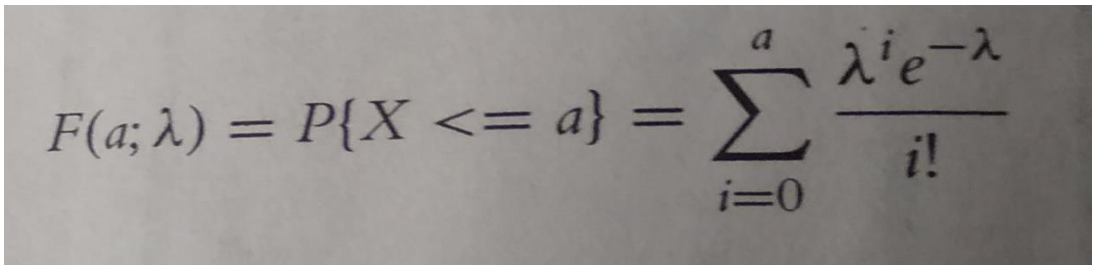
```
>qbinom(p=0.3,size=10,prob=0.3)
[1]  2
>qbinom(p=c(0.3,0.35,0.4,0.5,0.6), size=10,prob=0.3)
[1]  2  2  3  3  3
```

Poisson Distribution

- Another popular distribution is the Poisson Distribution, which is for count data. Its probability mass function is



$$p(x;\lambda) = \frac{\lambda^x e^{-\lambda}}{x!}$$



$$F(a;\lambda) = P\{X <= a\} = \sum_{i=0}^{a} \frac{\lambda^i e^{-\lambda}}{i!}$$

Where λ is both the mean and variance.

- To generate random counts, the density, the distribution and quantiles use rpois, dpois, ppois and qpois, respectively.
- As λ grows large the poisson distribution begins to resemble the normal distribution. To see this we will simulate 10,000 draws from the Poisson distribution and plot their histograms to see the shape.

```
>#generate 10,000 random counts from 5 different Poisson distributions
>pois1<-rpois(n=10000, lambda=1)
>pois2<-rpois(n=10000, lambda=2)
>pois5<-rpois(n=10000, lambda=5)
>pois10<-rpois(n=10000,lambda=10)
>pois20<-rpois(n=10000,lambda=20)
>pois<-data.frame(Lambda.1=pois1, Lambda.2=pois2, Lambda.5=pois5, Lambda.10=pois10, Lambda.20=pois20)
```

```
>#load reshape2 package to melt the data to make it easier to plot
>require(reshape2)
>#melt the data into a long format
>pois<-melt(data=pois, variable.name="Lambda", value.name="x")
>#load the stringr package to help clean up the new column name
>require(stringr)
>#clean up the lambda to just show the value for that lambda
>pois$Lambda<-as.factor(as.numeric(str_extract(string=pois$Lambda,
    pattern=\\d+)))
>head(pois)
  Lambda  x
1      1  0
2      1  2
3      1  0
4      1  1
5      1  2
6      1  0
```

```
>tail(pois)
       Lambda     x
49995  20         26
49996  20         14
49997  20         26
49998  20         22
49999  20         20
50000  20         23
>require(ggplot2)
>ggplot(pois,aes(x=x)) + geom_histogram(binwidth=1) + facet_wrap (~ Lambda) + ggtitle("Probability Mass Function")
```

- Another, perhaps more compelling, way to visualize this convergence to normality is within overlaid density plots.

```
>ggplot(pois, aes(x=x)) + geom_density(aes(group=Lambda, color=Lambda, fill=Lambda), adjust=4, alpha=1/2) + scale_color_discrete() + scale_fill_discrete() + ggtitle("Probability Mass Function")
```

Other Distributions

- R supports many distributions, some of which are very common, while others are quite obscure.

Statistical Distributions and their Functions

Distribution	Random Number	Density	Distribution	Quantile
Normal	rnorm	dnorm	pnorm	qnorm
Binomial	rbinom	dbinom	pbinom	qbinorm
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/Log-normal	rcauchy/rlnorm	dcauchy/dlnorm	pcauchy/plnorm	qcauchy/qlnorm
Multinomial/Logistic	rmultinom/rlogis	dmultinom/dlogis	pmultinom/plogis	qmultinom/qlogis
Hypergeometric	rhyper	dhyper	phyper	qhyper

Basic Statistics

- Some of the most common tools used in statistics are means, variances, correlations and t-tests. These are all well represented in R with easy-to-use-functions such as mean, var, cor and t.test.

Summary Statistics

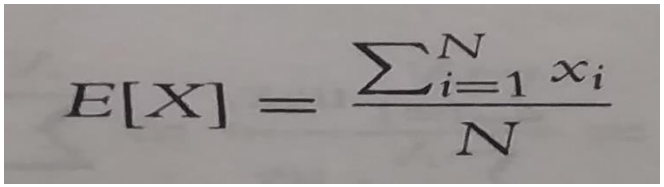
- The first thing many people think of in relation to statistics is the average, or mean, as it is properly called.

```
>x<-sample(x=1:100, size=100, replace= TRUE)
>x
[1]   81 100 79 66 32 87 85 10 46 76 71 25 100 51
[15] 17 60 15 6 60 89 23 100 64 28 74 12 12 10
[29] 48 79 10 87 50 21 96 85 40 41 64 18 92 70
[43] 44 28 73 22 35 93 18 64 43 50 53 52 84 99
[57] 25 61 41 47 1 66 93 22 77 56 17 74 19 85
[71] 25 11 27 83 100 37 53 64 60 55 69 58 62 53
[85] 62 36 96 68 11 54 45 69 35 63 38 16 95 1
[99] 80 69
```

- Sample uniformly draws size entries from x.
- Setting replace=TRUE means that the same number can be drawn multiple times.
- Now that we have a vector of data we can calculate the mean.

```
>mean(x)
[1] 53.17
```

- This is the simple arithmetic mean



- We need to consider cases where some data are missing. To create this we take x and randomly set 20% of the elements to NA.

```
>#copy x
>y<-x
>#choose a random 20 elements, using sample, to set to NA
>y[sample(x=1:100,size=20,replace=FALSE)]<-NA
>y
[1]   65 35 77 4 60 35 NA 55 75 78 63 11 71 77 NA 37 NA 90 69 19 NA 84 6 57
[25] 22 78 NA 41 NA NA 48 25 83 86 NA 13 NA 29 64 62 NA 88 54 84 71 36 40 NA
[49] 29 38 11 13 79 NA 59 6 4 9 50 12 NA 96 90 NA 85 43 82 NA 73 83 49 83
[73] 36 6 28 70 78 NA 35 99 NA 28 54 84 77 NA 27 NA NA 56 3 11 57 59 63 25
[97] 23 60 66 51
```

- Using mean on y will return NA. This is because, by default, if mean encounters even one element that is NA it will return NA. This is to avoid providing misleading information.

```
>mean(y)
[1] NA
```

- To have the NA's removed before calculating the mean, set na.rm to TRUE.

```
>mean(y,na.rm=TRUE)
[1] 51.025
```

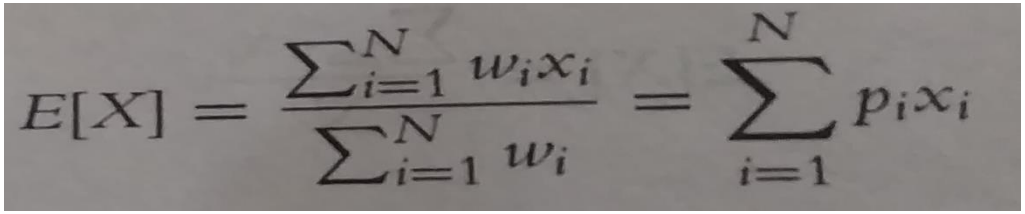
- To calculate the weighted mean of a set of numbers, the function weighted.mean takes a vector of numbers and a vector of weights. It also has an optional argument, na.rm, to remove NA's before calculating; otherwise, a vector with NA value will return NA.

```
>grades<-c(95,72,87,66)
>weights<-c(1/2,1/4,1/8,1/8)
>mean(grades)
[1] 80
```

>weighted.mean(x=grades,w=weights)

[1] 84.625

- The formula for weighted.mean is given by



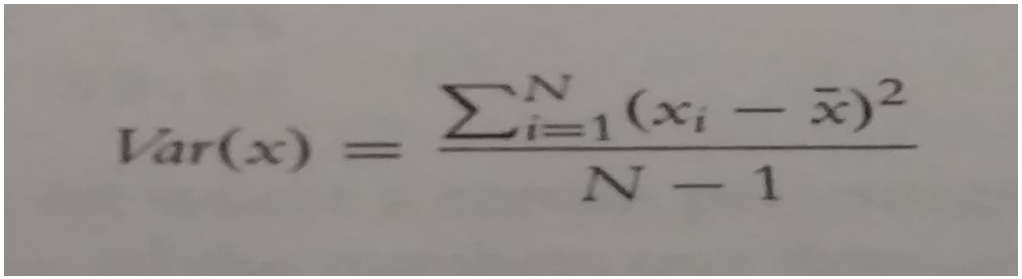
$$E[X] = \frac{\sum_{i=1}^{N} w_i x_i}{\sum_{i=1}^{N} w_i} = \sum_{i=1}^{N} p_i x_i$$

- Another vitally important metric is the variance, which is calculated with var.

>var(x)

[1] 710.0469

- This calculates variance as



$$Var(x) = \frac{\sum_{i=1}^{N} (x_i - \bar{x})^2}{N - 1}$$

Which can be verified in R.

>var(x)

[1] 710.0469

>sum((x-mean(x))^2)/(length(x)-1)

[1] 710.0469

- Standard deviation is the square root of variance and is calculated with sd. Like mean and var, sd has the na.rm argument to remove Nas before computation; otherwise, any NAS will cause the answer to be NA.

```
>sqrt(var(x))
[1] 26.6467
>sd(x)
[1] 26.6467
>sd(y)
[1] NA
>sd(y,na.rm= TRUE)
[1] 27.4761
```

- Other commonly used functions summary statistics are min, max and median. Of course all of these also have na.rm argumenys.

```
 >min(x)
[1]  3
>max(x)
[1]  99
>median(x)
[1]  43
>min(y)
```

>min(y, na.rm=TRUE)

[1] 2

Note: the median, as calculated before, is the middle of an ordered set of numbers. For instance, the median of 5,2,1,8 and 6 is 5. In this case where there is an even amount of numbers, the median is the mean of the middle two numbers. For 5,1,7,4,3,8,6 and 2 the median is 4.5.

- A helpful function that computes the mean, minimum, maximum and median is summary. There is no need to specify na.rm because if there are NA's, they are automatically removed and their count is included in the results.

>summary(x)

Min	1st Qu.	Median	Mean	3rd Qu.	Max
1.00	17.75	43.00	44.51	68.25	100.00

>summary(y)

Min	1st Qu.	Median	Mean	3rd Qu.	Max	NA's
2.00	18.00	40.50	43.59	67.00	100.00	20

- This summary also displayed the first and third quantiles. These can be computed using quantile.

>#calculate the 25th and 75th quantile

>quantile(x,probs=c(0.25,0.75))

25% 75%

17.75 68.25

>#try the same on y

>quantile(y, probs=c(0.25,0.75))

Error: missing values and NaN's not allowed if 'na.rm' is FALSE

>#this time use na.rm=TRUE

>quantile(y, probs=c(0.25,0.75),na.rm=TRUE)

25% 75%

18 67

>#compute other quantiles

Quantile(x, probs=c(0.1,0.25,0.5,0.75,0.99))

10% 25% 50% 75% 100%

6.00 12.75 43.00 68.25 98.02

Note: Quantiles are numbers in a set where a certain percentage of the numbers are smaller than the quantile. For instance, of the numbers one through 200, the 75th quantile the number that is larger than 75% of the

Correlation and Covariance

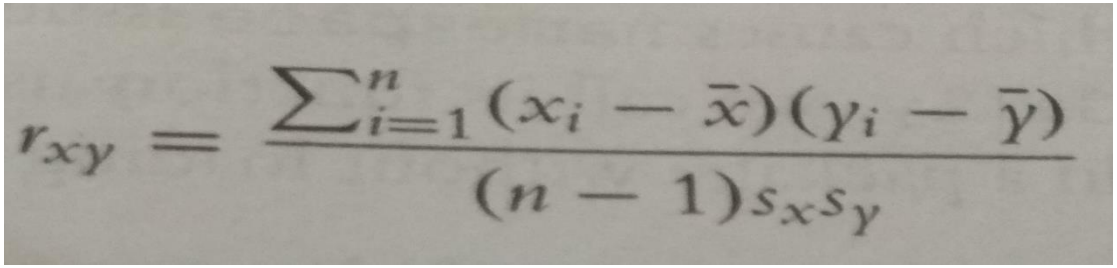
- When dealing with more than one variable, we need to test their relationships with each other.
- Two simple, straightforward methods are correlation and covariance. To examine these concepts we look at the economics data from ggplot2.

```
>require(ggplot2)
>head(economics)
```

	date	pce	pop	psavert	uempmed	unemploy
	<date>		*<dbl>*	*<int>*	*<dbl>*	*<dbl>*	*<int>*
1	1967-07-01	507.	198712	12.5	4.5	2944	
2	1967-08-01	510.	198911	12.5	4.7	2945	
3	1967-09-01	516.	199113	11.7	4.6	2958	
4	1967-10-01	513.	199311	12.5	4.9	3143	
5	1967-11-01	518.	199498	12.5	4.7	3066	
6	1967-12-01	526.	199657	12.1	4.8	3018	

```
>cor(economics$pcs,economics$psavert)
[1]  -0.9271222
```

- This very low correlation makes sense because spending and saving are opposites of each other. Correlation is defined as



$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{(n-1)s_x s_y}$$

```
>#use cor to calculate correlation
>cor(economics$pce, economics$psavert)
[1] -0.837069
>##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] -0.837069
```

- To compare multiple variables at once, use cor on a matrix(only for numeric variables).

```
>cor(economics[ , c(2, 4:6)])
                pce        psavert        uempmed        unemploy
pce        1.0000000 -0.8370690  0.7273492  0.6139997
psavert   -0.8370690  1.0000000 -0.3874159 -0.3540073
uempmed    0.7273492 -0.3874159  1.0000000  0.8694063
unemploy   0.6139997 -0.3540073  0.8694063  1.0000000
```

- Because this is just a table of numbers, it would be helpful to also visualize the information using a plot.
- For this we use the ggpairs function from the Ggally package (a collection of helpful plots built on ggplot2). This shows a scatterplot of every variable in the data against every other variable.
- Loading GGally also loads the reshape package, which causes namespace issues with the newer reshape2 package. So, rather than load GGally, we call its function using the :: operator, which allows access to functions within a package without loading it.

```
>Ggally::ggpairs(economics[ , c(2,4:6)],
   params=list[labelSize=8))
```

- This is similar to a small multiples plot except that each pane has different x- and y- axes. While this shows the original data, it doesnot actually show the correlation numbers.
- High positive correlation indicates a positive relationship between the variables, high negative correlation indicates a negative relationship between the variables and near zero correlation indicates no strong relationship.

```
>#load the reshape package for melting the data
>require(reshape2)
>#load the scales package for some extra plotting features.
>require(scales)
>#build the correlation matix
>econCor<-cor(economics[ , c(2,4:6)])
```

- >#melt it into the long format

```
>econMelt<-melt(econCor, varnames=c("x", "y"), value.name="Correlation")
>#order it according to the correlation
>econMelt<-econMelt(order(econMelt$Correlation), ]
>#display the melted data
>econMelt
```

	x	y	Correlation
2	psavert	pce	-0.8370690
5	pce	psavert	-0.8370690
7	uempmed	psavert	-0.3874159
10	psavert	uempmed	-0.3874159
8	unemploy	psavert	-0.3540073
14	psavert	unemploy	-0.3540073
4	unemploy	pce	0.6139997
13	pce	unemploy	0.6139997
3	uempmed	pce	0.7273492
9	pce	uempmed	0.7273492
12	unemploy	uempmed	0.8694063
15	uempmed	unemploy	0.8694063
1	pce	pce	1.0000000
6	Psavert	psavert	1.0000000
11	uempmed	uempmed	1.0000000
16	unemploy	unemploy	1.0000000

```
>#  plot it with ggplot
>#initialize the plot with x and y on the x and y axes
>ggplot(econMelt, aes(x=x,y=y)) +
+    #draw tiles filling the color based on Correlation
> geom_tile(aes(fill=Correlation))+
+   #make the fill (color) scale a three color gradient with muted
+   #red for the low point, white for the middle and steel blue
+   #for the high point
+   #the guide should be a colorbar with no ticks, whose height is
+   #10 lines
+   #limits indicates the scale should be filled from -1 to 1
+   scale_fill_gradient2(low=muted("red"),mid="white",
     high="steelblue",guide=guide_colorbar(ticks=FALSE,
     barheight=10), limits=c(-1,1)) + theme_minimal() + labs(x=NULL,
     y=NULL)
```

•Missing data is just as much a problem with cor as it is with mean and var, but it is dealt with differently because multiple columns are being considered simultaneously.

•Instead of specifying na.rm=TRUE to remove NA entries, one of “all.obs”, “complete.obs”,”pairwise.complete.obs”,”everything” or “na.or.complete” is used.

•To illustrate this we first make a five-column matrix where only the fourth and fifth columns have no NA values; the other columns have one or two Nas.

```
>m<-c(9,9,NA,3,NA,5,8,1,10,4)
>n<-c(2,NA,1,6,6,4,1,1,6,7)
>p<-c(8,4,3,9,10,NA,3,NA,9,9)
>q<-c(10,10,7,8,4,2,8,5,5,2)
>r<-c(1,9,7,6,5,6,2,7,9,10)
>#combine them together
>theMat<-cbind(m,n,p,q,r)
```

- The first option for use is “everything”, which means that the entirety of all columns must be free of NA’s, otherwise the result is NA. Running this should generate a matrix of all Nas except ones on the diagonal- because a vector is always perfectly correlated with itself- and between q and r.
- With the second option- “all.obs”-even a single NA in any column will cause an error.

>cor(theMat, use="everything")

	m	n	p	q	r
m	1	NA	NA	NA	NA
n	NA	1	NA	NA	NA
p	NA	NA	1	NA	NA
q	NA	NA	NA	1.0000000	-0.4242958
R	NA	NA	NA	-0.4242958	1.0000000

>cor(theMat,use="all.obs")

Error in cor(theMat, use = "all.obs") : missing observations in cov/cor

- The third and fourth options- "complete.obs" and "na.or.complete"- work similarly to each other in that they keep only rows where every entry is not NA.
- That means our matrix will be reduced to rows 1,4,7,9 and 10, and then have its correlation computed.
- The difference is that "complete.obs" will return an error if not a single complete row can be found, while "na.or.complete" will return NA in that case.

>cor(theMat,use="complete.obs")

	m	n	p	q	r
m	1.0000000	-0.5228840	-0.2893527	0.2974398	-0.3459470
n	-0.5228840	1.0000000	0.8090195	-0.7448453	0.9350718
p	-0.2893527	0.8090195	1.0000000	-0.3613720	0.6221470
q	0.2974398	-0.7448453	-0.3613720	1.0000000	-0.9059384
r	-0.3459470	0.9350718	0.6221470	-0.9059384	1.0000000

```
>cor(theMat,use="na.or.complete")
          m         n         p         q         r
m 1.0000000 -0.5228840 -0.2893527 0.2974398 -0.3459470
n -0.5228840 1.0000000 0.8090195 -0.7448453 0.9350718
P -0.2893527 0.8090195 1.0000000 -0.3613720 0.6221470
q 0.2974398 -0.7448453 -0.3613720 1.0000000 -0.9059384
r -0.3459470 0.9350718 0.6221470 -0.9059384 1.0000000
>#calculate the correlation just on complete rows
>cor(theMat[c(1,4,7,9,10), ])
          m         n         p         q         r
m 1.0000000 -0.5228840 -0.2893527 0.2974398 -0.3459470
n -0.5228840 1.0000000 0.8090195 -0.7448453 0.9350718
p -0.2893527 0.8090195 1.0000000 -0.3613720 0.6221470
q 0.2974398 -0.7448453 -0.3613720 1.0000000 -0.9059384
r -0.3459470 0.9350718 0.6221470 -0.9059384 1.0000000

>#compare "complete.obs" and computing on select rows
>#should give the same result
>identical(cor(theMat, use="complete.obs"), cor(theMat[c(1,4,7,9,10),
    ])
[1] TRUE
```

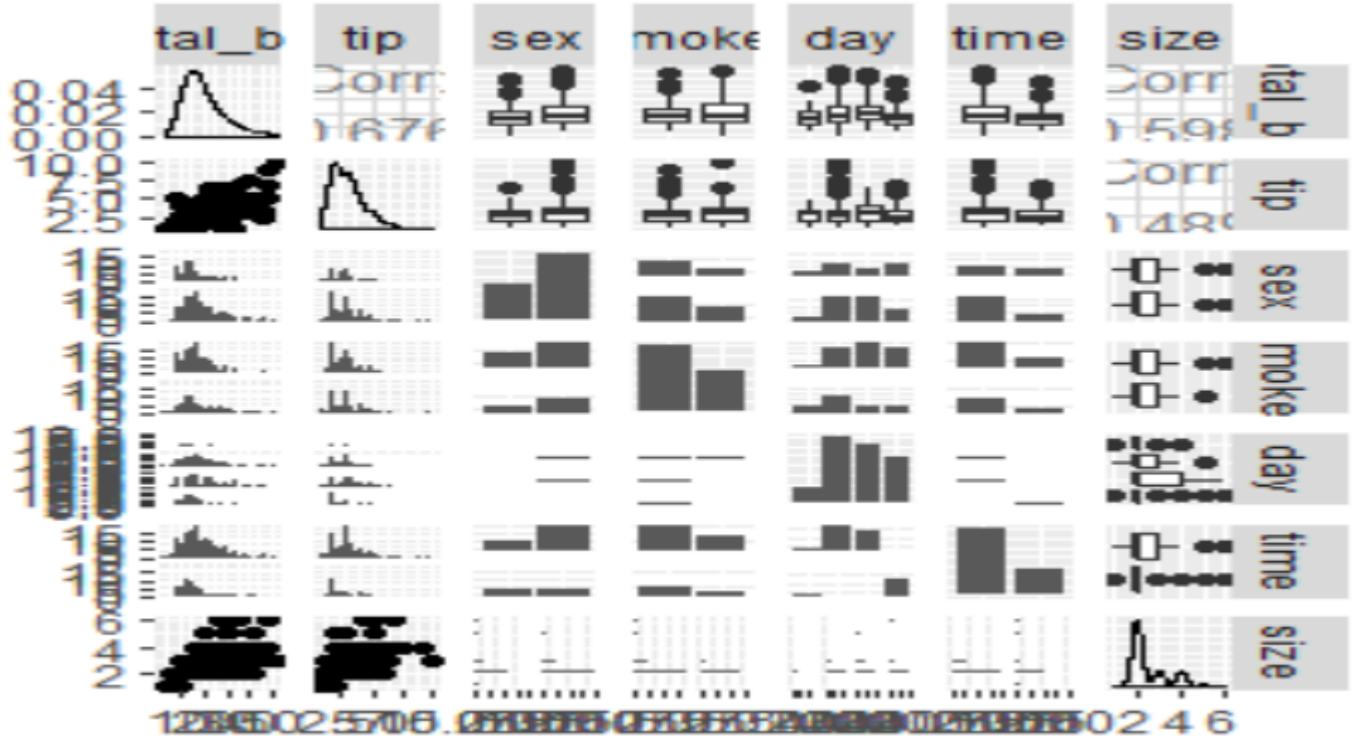
- The final option is “pairwise.complete”, which is much more inclusive. It compares two columns at a time and keep rows- for those two columns- where neither entry is NA.
- This is essentially the same as computing the correlation between every combination of two columns with use set to “complete.obs”.

>#the entire correlation matrix

>cor(theMat,use=“pairwise.complete.obs”)

	m	n	p	q	r
m	1.00000000	-0.02511812	-0.3965859	0.4622943	-0.2001722
n	-0.02511812	1.00000000	0.8717389	-0.5070416	0.5332259
p	-0.39658588	0.87173889	1.0000000	-0.5197292	0.1312506
q	0.46229434	-0.50704163	-0.5197292	1.0000000	-0.4242958
r	-0.20017222	0.53322585	0.1312506	-0.4242958	1.0000000

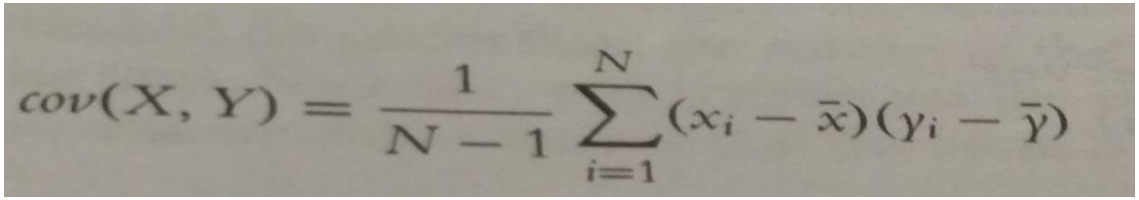
```
>#compare the entries for m vs n to this matrix
>cor(theMat[, c("m","n")], use="complete.obs")
                          m                    n
m                 1.00000000          -0.02511812
n                -0.02511812           1.00000000
>#compare the entries for m vs p to this matrix
>cor(theMat[ , c("m","p")], use ="complete.obs")
                          m                    p
m                 1.0000000          -0.3965859
p                -0.3965859           1.0000000
>data(tips,package="reshape2")
>head(tips)
     total_bil        tip          sex       smoker     day        time        size
1    16.99            1.01         Female       No      Sun        Dinner       2
2    10.34            1.66          Male        No      Sun        Dinner       3
3    21.01            3.50          Male        No      Sun        Dinner       3
4    23.68            3.31          Male        No      Sun        Dinner       2
5    24.59            3.61         Female       No      Sun        Dinner       4
6    25.29            4.71          Male        No      Sun        Dinner       4
>GGally:: ggpairs(tips)
```


tal_b
tip
sex
moke
day
time
size

>require(RXKCD)

>getXKCD(which="552")

- Similar to correlation is covariance, which is like a variance between variables;



$$cov(X, Y) = \frac{1}{N-1} \sum_{i=1}^{N} (x_i - \bar{x})(y_i - \bar{y})$$

- The cov function works similarly to the cor function, with the same arguments for dealing with missing data.

>cov(economics$pce,economics$psavert)

[1] -9361.028

>cov(economics[, c(2,4:6)])

	pce	psavert	unempmed	unemploy
pce	12811296.900	-9361.028324	10695.02387	5806187.162
psavert	-9361.028	9.761835	-4.972622	-2922.162
uempmed	10695.024	-4.972622	16.876582	9436.074
unemploy	5806187.162	-2922.161618	9436.074287	697955.661

```
>#check that cov and cor*sd*sd are the same
>identical(cov(economics$pce,economics$psavert,cor(economics$pce,econom
    ics$psavert)*sd(economics$pce)*sd(economics$psavert))
[1]  TRUE
```

T-Tests

- In a traditional statistics classes, the t-test – invented by William Gosset while working at the Guinness brewery- is taught for conducting tests on the mean of data or for comparing two sets of data.

```
>head(tips)
```

	total_bil	tip	sex	smoker	day	time	size
1	16.99	1.01	Female	No	Sun	Dinner	2
2	10.34	1.66	Male	No	Sun	Dinner	3
3	21.01	3.50	Male	No	Sun	Dinner	3
4	23.68	3.31	Male	No	Sun	Dinner	2
5	24.59	3.61	Female	No	Sun	Dinner	4
6	25.29	4.71	Male	No	Sun	Dinner	4

```
>#sex of the server
>unique(tips$sex)
[1]  Female  Male
Levels:  Female  Male
>#day of the week
>unique(tips$day)
[1] Sun Sat Thur Fri
 Levels: Fri Sat Sun Thur
```

One-Sample T-Test

- First we conduct a one-sample t-test on whether the average tip is equal to $2.50.
- This text essentially calculates the mean of data and builds a confidence interval. If the value we are testing falls within that confidence interval then we can conclude that is the true value for the mean of the data; otherwise, we conclude that it is not the true mean.

```
>t.test(tips$tip,alternative="two.sided", mu=2.5)
```

One Sample t-test

data: tips$tip

t = 5.6253, df = 243, p-value = 5.08e-08

alternative hypothesis: true mean is not equal to 2.5

95 percent confidence interval:

2.823799 3.172758

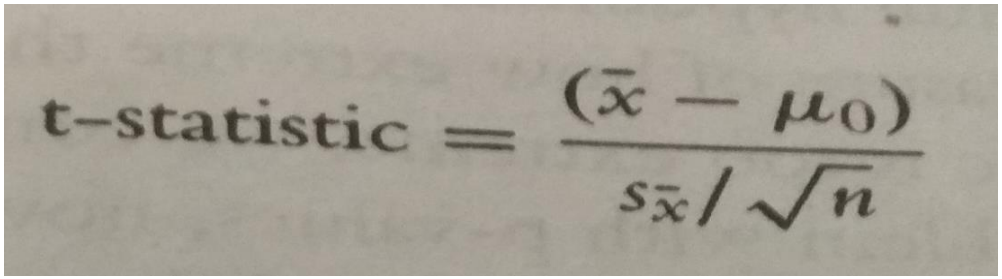
sample estimates:

mean of x

2.998279

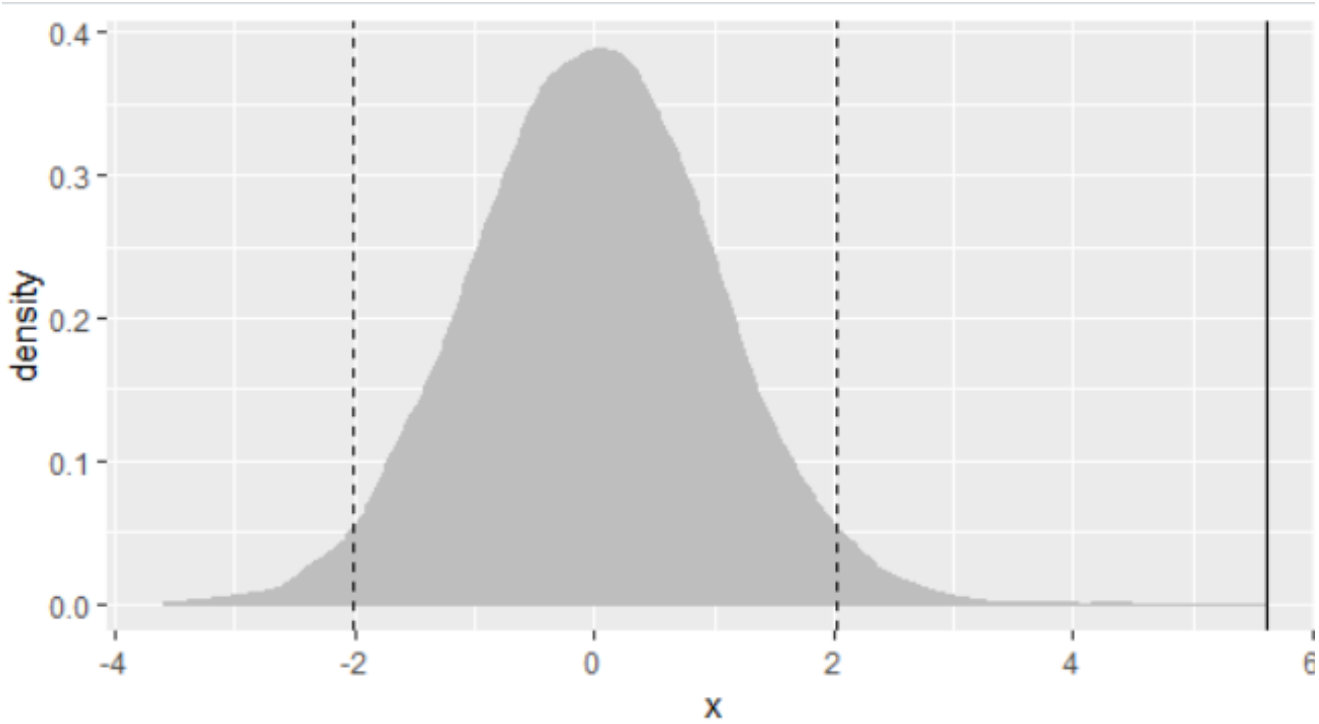
- The output displays the setup and results of the hypothesis test of whether the mean is equal to $2.50.
- It prints the t-statistic, the degree of freedom and p-value.
- It also provides the 95% confidence interval and mean for the variable of interest.
- The p value indicates the null hypothesis should be rejected, and we conclude that the mean is not equal to $2.50.

- The t-test is the ratio where the numerator is the difference between the estimated mean and the hypothesized mean and the denominator is the standard error of the estimated mean. It is defined as



- If the hypothesized mean is correct, then we expect the t-statistic to fall somewhere in the middle- about two standard deviations from the mean- of the t distribution.

```
>##build a t distribution
>randT<-rt(30000, df=NROW(tips)-1)
>#get t-statistics and other information
>tipTTest<-t.test(tips$tip, alternative="two.sided", mu=2.50)
>#plot it
>ggplot(data.frame(x=randT))+geom_density(aes(x=x),
    fill="gry",color="grey")+geom_vline(xintercept=tipTest$statistic)+
    geom_vline(xintercept=mean(randT)+c(-2,2)*sd(randT),
    linetype=2)
```



T distribution and t-statistic for tip data. The dashed lines are two standard deviations from the mean in either direction. The thick black line, the t-statistic, is so far outside the distribution that we must reject the null hypothesis and conclude that the true mean is not $2.50.

- Conduction of one-sided t-test to see if the mean is greater than $2.50

```
>t.test(tips$tip,alternative="greater",mu=2.5)
```

Output:

```
One Sample t-test
data: tips$tip
t = 5.6253, df = 243, p-value = 2.54e-08
alternative hypothesis: true mean is greater than 2.5
95 percent confidence interval:
2.852023      Inf
sample estimates:
mean of x
2.998279
```

- Once again, the p-value indicates that we should reject the null hypothesis and conclude that the mean is greater than $2.50, which coincides nicely with the confidence interval.

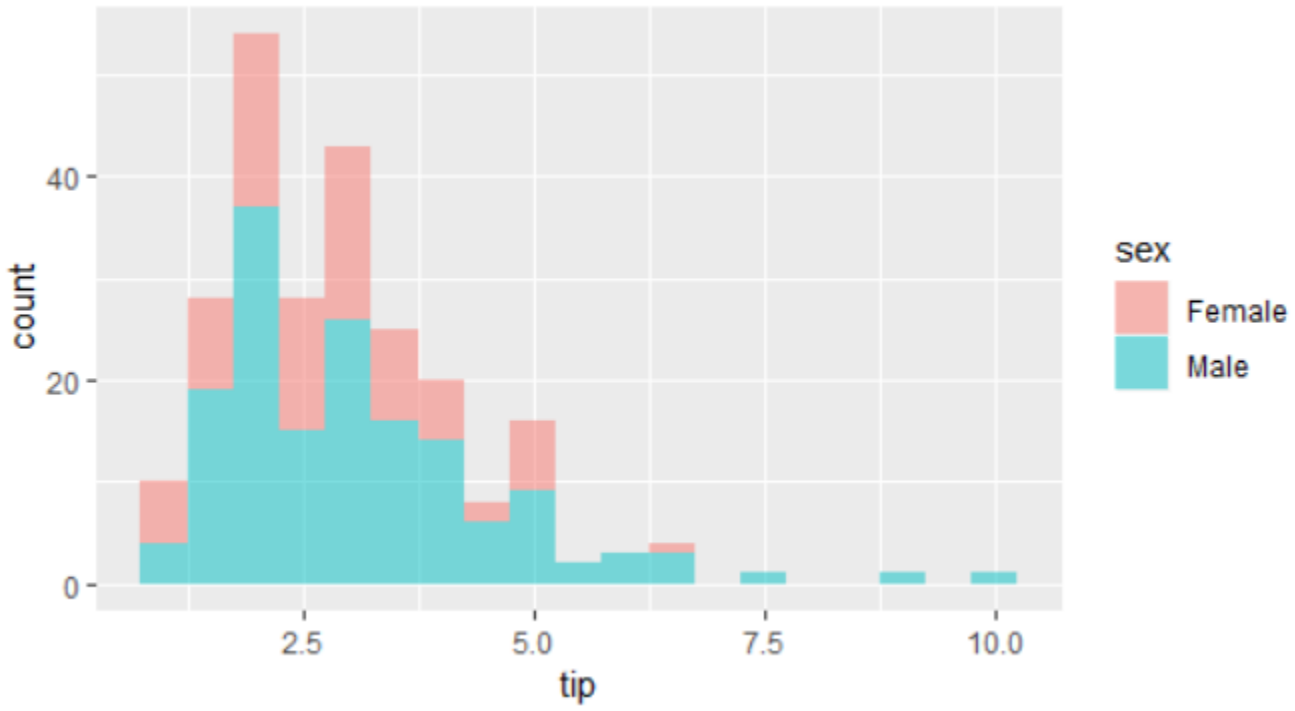
Two-Sample T-Test:

- More often or not the t-test is used for comparing two samples.
- Continuing with the tips data, we compare how female and male servers are tipped. Before running the t-test, however, we first need to check the variance, whereas the Welch two-sample t-test can handle groups with differing variances.

```
>#first just compute the variance for each group;
>#using the formula interface
>#calculate the variance of tip for each level of sex
>aggregate(tip~sex,data=tips,var)
     sex      tip
1 Female 1.344428
2   Male 2.217424
>#now test for normality of tip distribution
>shapiro.test(tips$tip)
```

Output:

```
Shapiro-Wilk normality test
data:  tips$tip
W = 0.89781, p-value = 8.2e-12
```



Histogram of tip amount by sex.

- Since the data do not appear to be normally distributed, neither the standard F-test nor the Bartlett test will suffice. So we use the nonparametric Ansari-Bradley test to examine the equality of variances.

```
>ansari.test(tip~sex, tips)
```

Output:

```
Ansari-Bradley test
data: tip by sex
AB = 5582.5, p-value = 0.376
alternative hypothesis: true ratio of scales is not equal to 1
```

- This test indicates that the variances are equal, meaning we can use the standard two-sample t-test.

```
>#setting var.equal=TRUE runs a standard two sample t-test whereas
>#var.equal=FALSE (the default) would run the Welch test
>t.test(tip~sex, data=tips, var.equal=TRUE)
```

Output

```
Two Sample t-test
data: tip by sex
t = -1.3879, df = 242, p-value = 0.1665
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
-0.6197558 0.1074167
sample estimates:
mean in group Female          mean in group Male
 2.833448                      3.089618
```

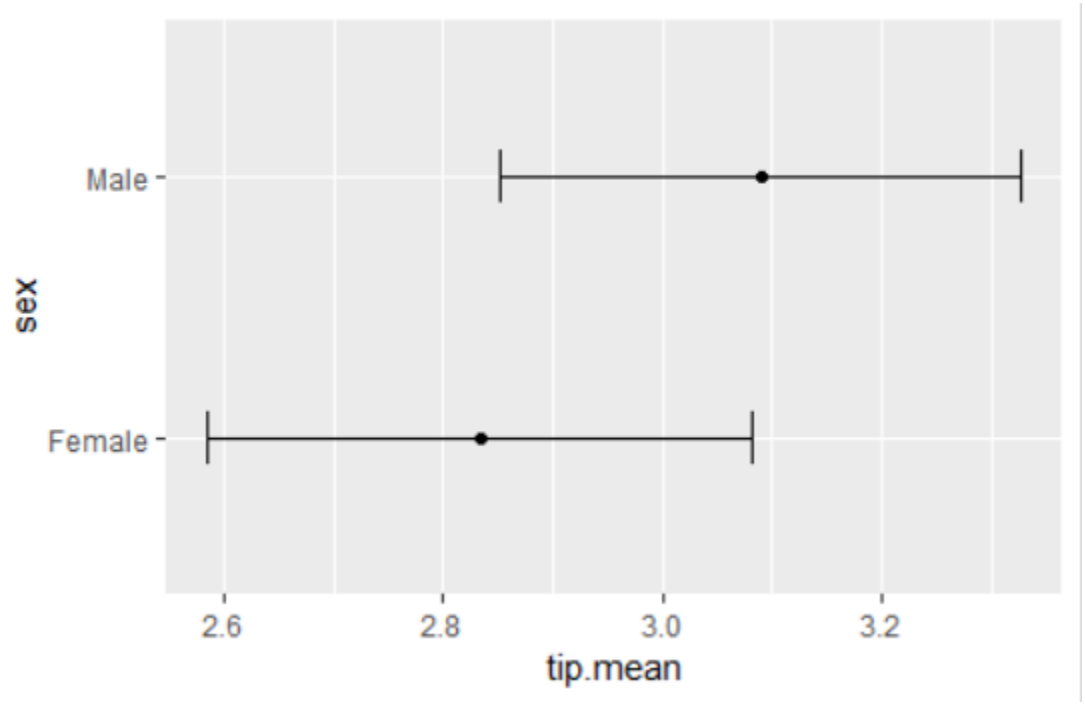
```
>require(plyr)
>tipSummary<-ddply(tips,
    "sex",summarize,tip.mean=mean(tip),tip.sd=sd(tip),lower=tip.mean-
    2*tip.sd/sqrt(NROW(tip)),Upper=tip.mean+2*tip.sd/sqrt(NROW(tip)))
>tipSummary
```

Output:

	sex	tip.mean	tip.sd	lower	Upper
1	Female	2.833448	1.159495	2.584827	3.082070
2	Male	3.089618	1.489102	2.851931	3.327304

- ddply was used to split the data according to the levels of sex. It then applied the summarize function to each subset of the data. This function applied the indicated functions of the data, creating a new data.frame.
- We prefer visualizing the results rather than comparing numerical values. This requires reshaping the data a bit.

```
>ggplot(tipSummary,aes(x=tip.mean,y=sex))+geom_point()+geom_errorbarh(
    aes(xmin=lower,xmax=Upper),height=.2)
```



Plot showing the mean and two standard errors of tips broken down by the sex of the server.

Paired Two-Sample T-Test

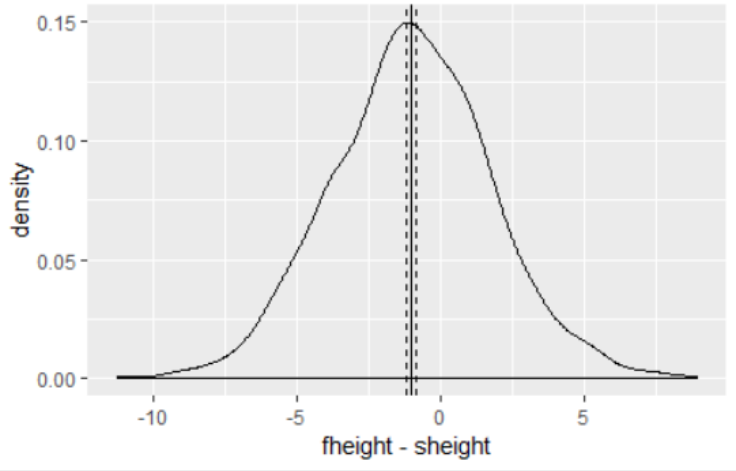
• For testing paired data a paired t-test should be used. This is simple enough to do by setting the paired argument in t.test to TRUE.

• Heights are generally normally distributed, so we will forgo the tests of normality and equal variance.

```
> require(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
>t.test(father.son$fheight, father.son$sheight, paired=TRUE)
Paired t-test
data: father.son$fheight and father.son$sheight
t = -11.789, df = 1077, p-value < 2.2e-16
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -1.1629160     -0.8310296
sample estimates:
mean of the differences
 -0.9969728
```

- The test shows that we should reject the null hypothesis and conclude that fathers and sons have different heights.
- We visualize this data using a density plot of the differences. In it we see a distribution with a mean not at zero and a confidence interval that barely excludes zero which agrees with the test.

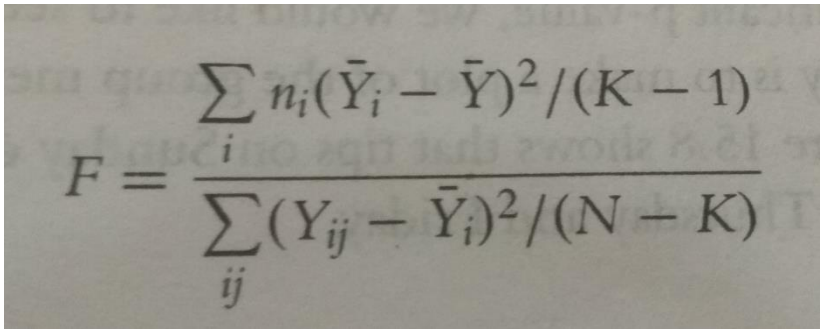
```
>heightDiff<-father.son$fheight-father.son$sheight
>ggplot(father.son,aes(x=fheight-
    sheight))+geom_density()+geom_vline(xintercept =
    mean(heightDiff))+geom_vline(xintercept = mean(heightDiff)+2*c(-
    1,1)*sd(heightDiff)/sqrt(nrow(father.son)),linetype=2)
```



Density plot showing the difference of heights of fathers and sons

ANOVA

- After comparing two groups, the natural next step is comparing multiple groups. Every year, far too many students in introductory statistic classes are forced to learn the ANOVA(analysis of variance) test and memorize its formula, which is



$$F = \frac{\sum_i n_i(\bar{Y}_i - \bar{Y})^2/(K-1)}{\sum_{ij}(Y_{ij} - \bar{Y}_i)^2/(N-K)}$$

```
>tipAnova<-aov(tip~day-1,tips)
tipIntercept<-aov(tip~day,tips)
tipAnova$coefficients
Output:
          dayFri      daySat     daySun     dayThur
          2.734737 2.993103 3.255132   2.771452
>tipIntercept$coefficients
Output
 (Intercept)         daySat          daySun          dayThur
 2.73473684      0.25836661      0.52039474      0.03671477
```

```
>summary(tipAnova)
                Df    Sum Sq   Mean Sq  F value Pr(>F)
day              4    2203.0     550.8    290.1 <2e-16 ***
Residuals      240     455.7       1.9
---
Signif. codes: 0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
>tipsByDay<-
    ddply(tips,"day",summarize,tip.mean=mean(tip),tip.sd=sd(tip),length=NROW(tip),tfrac=
    qt(p=.90,df=length-1),Lower=tip.mean-
    tfrac*tip.sd/sqrt(Length),Upper=tip.mean+tfrac*tip.sd/sqrt(Length))
>ggplot(tipsByDay,aes(x=tip.mean,y=day))+geom_point()+geom_errorbarh(aes(xmin=Lower
    , xmax=Upper), height=.3)
```

- The use of NROW instead of nrow is to guarantee computation. Where nrow works only on data.frames and matrices, NROW returns the length of objects that have only one dimension.

```
>nrow(tips)
[1]  244
>NROW(tips)
[1]  244
>nrow(tips$tip)
NULL
>NROW(tips$tip)
[1]  244
```