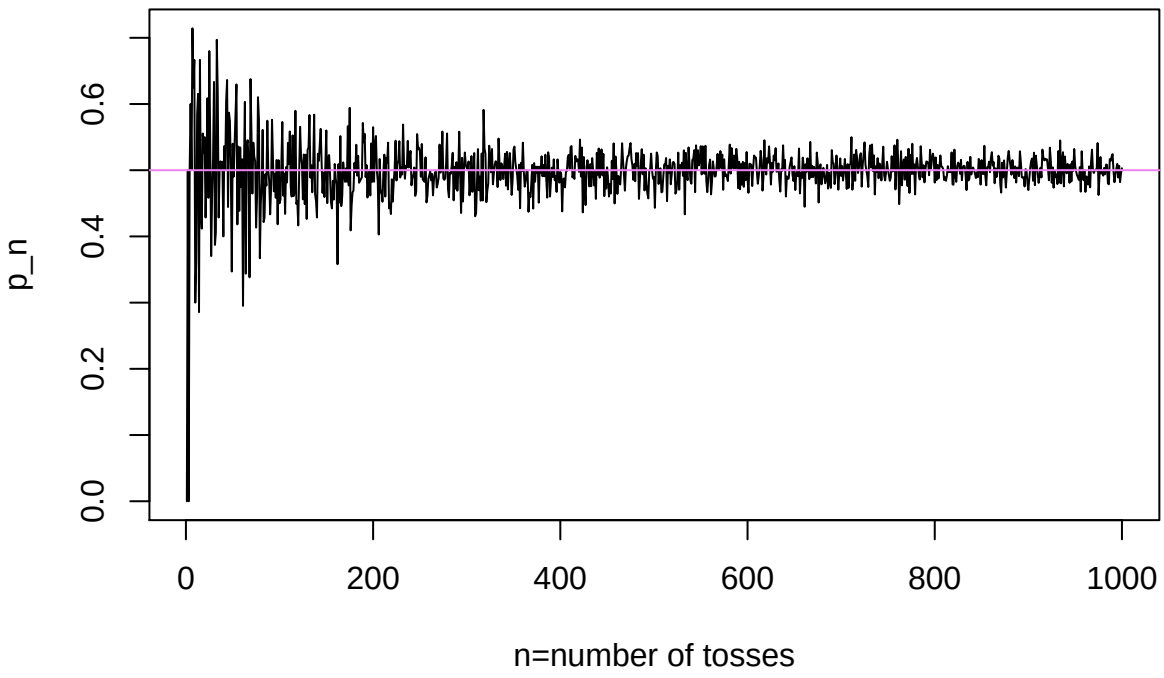
Intro to R for MATH 109: Scripts from Class

All of the following code can be found on Blackboard in R script files.

Law of Large Numbers Simulation

This code demonstrates the Law of Large Numbers with a simulated coin toss. The function `function` is a way to create a function that doesn't already exist in basic R or a supplimentary package.

```
# This coin toss simulation illustrates the Law of Large Numbers
# create a function to simulate a n coin tosses (Heads=1, Tails=0) of a fair coin
# and compute the mean of the tosses
coin.toss = function(n) {
      pn=mean(rbinom(n,1,.5))
      return(pn)
}
# any time we use a randomization like rbinom, the numbers are generated randomly
# you can set a seed here so random results are replicable
set.seed(100)
# set the number of tosses
n=1000
# create a vector storing the means of coin tosses 1:n
p <- 0
for (i in 1:n) {
  p[i] <- coin.toss(i)
}
# plot the tosses
plot(p, type = "l", xlab = "n=number of tosses", ylab="p_n")
# add a line showing the probability of tossing heads
abline(.5,0,col='violet',lty=1, lwd=1)
```


p_n
0.0
0.2
0.4
0.6
0
200
400
600
800
1000
n=number of tosses

Permutation Test Example

This code demonstrates the permutation test. It can easily be modified for an appropriate data set.

```
# Using the built in data set MT Cars, we will test if number of cylinders effects mpg
# So H0=mu1=mu2 (null hypothesis: cylinders has no impact on mpg)
# where mu1 is the mpg mean for 4 cyl,  mu2 is the mpg mean for 6 cyl
#(this dataset also includes 8 cyl cars, but we will only make the one comparison)
# attach the data set
attach(mtcars)

# change number of cylinders from numeric to factor
as.factor(cyl)
```

```
##  [1] 6 6 4 6 8 6 8 4 4 6 6 8 8 8 8 8 8 4 4 4 4 8 8 8 8 4 4 4 8 6 8 4
## Levels: 4 6 8
```

```
# create new data frame with the wanted values, "!" means "not"
# We don't care about 8 cylinder cars
new.mt=mtcars[mtcars$cyl!="8", 1:2]

# find the mean mpg for each of the cylinder types
# we use the assignment "x="
# using ";" allows us to run another command without taking up another line
# note that R (and most programming languages are case sensitive)
# the "==" is a logical assignment, so we want to look at cars with a certain cylinder assignment
m1=mean(new.mt[new.mt$cyl=="4",1]);m1
```

```
## [1] 26.66364
```

```
m2=mean(new.mt[new.mt$cyl=="6",1]);m2
```

```
## [1] 19.74286
```

```
# find the point estimates for the differences in mpg means
point.estimate=m1-m2;point.estimate
```

```
## [1] 6.920779
```

```
# now we will run a permutation test
# set the seed so the results can be repeated
set.seed(100)

# randomly split the mpg values into two groups
perm = sample(1:2,size=nrow(new.mt),replace=TRUE,prob=c(1/2,1/2))

# simulation results (means of mpg) where any difference in mpg is due to chance
p1=mean(new.mt[perm==1,1])
p2=mean(new.mt[perm==2,1])

# create a data frame to compare mpg averages
m = data.frame("4cyl" = c(m1,p1), "6cyl" = c(m2,p2));m
```

```
##      X4cyl    X6cyl
## 1 26.66364 19.74286
## 2 26.30000 22.49091
```

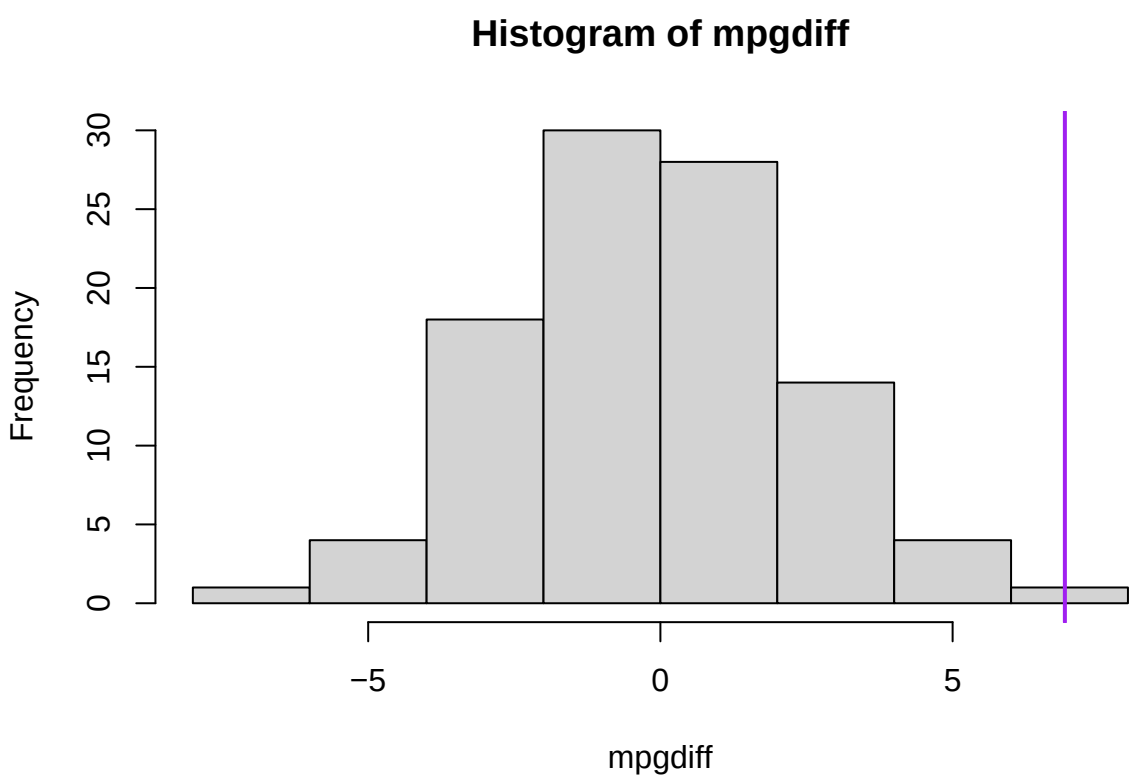
```
# calculate the differences to compare with the point estimates
diff=p1-p2;diff
```

```
## [1] 3.809091
```

```
point.estimate
```

```
## [1] 6.920779
```

```
# we can repeat the permutation again to see what happens

# now we want to repeat this n times and store the differences
# Use the replicate function to create n permutations
n=100
x = replicate(n, {
  permute = sample(1:2,size=nrow(new.mt),replace=TRUE,prob=c(1/2,1/2))
})
# create a function to find the means at each index of permutation (i) for each category (m)
mpg.means = function(i,m) {
  mpgmean=mean(mtcars[x[,i]==m,1])
  return(mpgmean)
}
# store the mpg means
mpgmean1 <- 0
for (i in 1:n) {
  mpgmean1[i] <- mpg.means(i,1)
}
mpgmean2 <- 0
for (i in 1:n) {
  mpgmean2[i] <- mpg.means(i,2)
}

# compute the difference in mpg means
mpgdiff=mpgmean1-mpgmean2

# compute the mean of the differnce
permmean=mean(mpgdiff)

#look at a histogram of the differences in the sample vs the point estimate
hist(mpgdiff)
abline(v=point.estimate, lwd=2, col="purple")
```



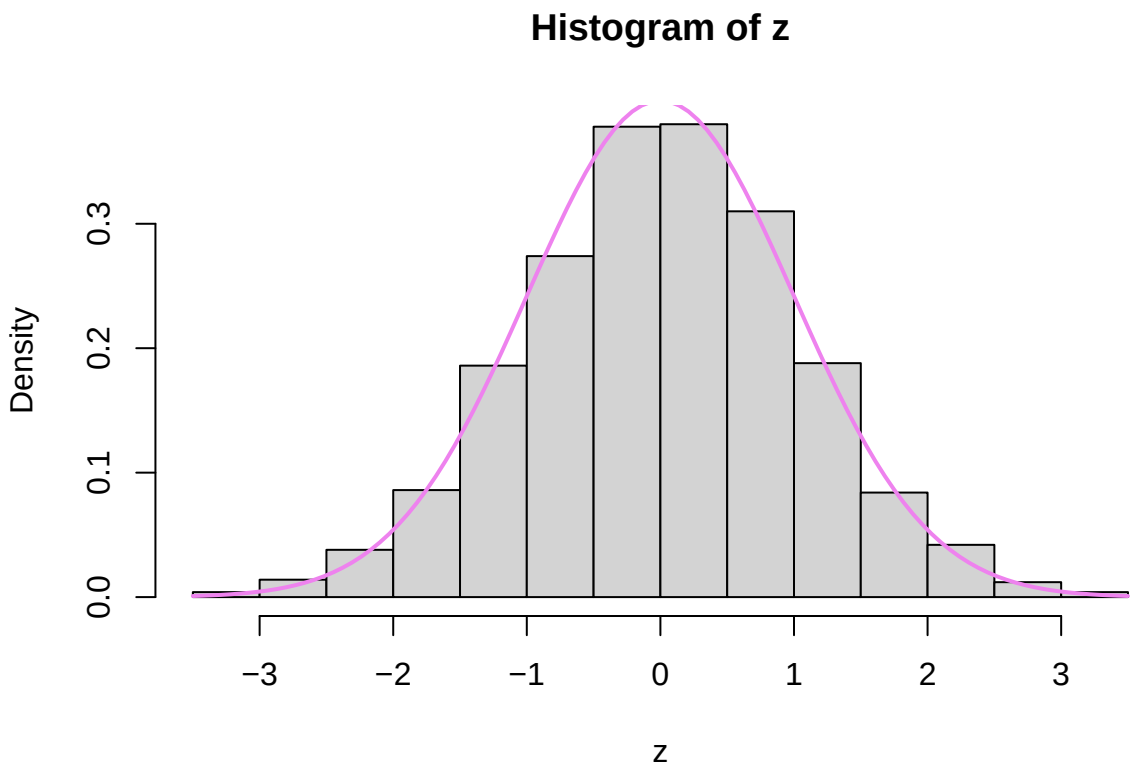

```
# find the p-value
# find the number of times the absolute value of
# the permuted difference equaled or exceeded the point estimate
x=sum(ifelse(mpgdiff>=abs(point.estimate),1,0))
p.val=x/n;p.val
```

```
## [1] 0
```

Central Limit Theorem Simulation

This code simulates the Central Limit Theorem by picking the number of independent observations/samples, as well as pick a mean and variance to adjust the center and spread of the resulting histogram.

```
# CLT simulation
set.seed(100)
n=1000 # set the number of independent observations
m=0 # set the sample mean
s=1 # set the sample standard deviation
b=100 # set the bin width on the histogram - in R it's called "breaks"
z=rnorm(n,m,s) # this generates n samples drawn according to a normal dist with mean m and sd s
# create a probability histogram that shows the distribution of the generated observations
# hist default is a frequency histogram
hist(z, prob=TRUE)
# draw the normal curve for comparison
curve(dnorm(x,mean=m,sd=s),add=TRUE,lwd=2,col="violet")
```



Confidence Interval Function (Using $z^\star$)

This code can automatically compute a confidence interval when given the point estimate, confidence level in decimal form and standard error.

```
# function for any size confidence interval
conf.int=function(p=point.estimate,c=confidence.level,s=standard.error){
  CI.l=p-qnorm(1-(1-c)/2)*s  # this computes the lower bound for the confidence interval
  CI.u=p+qnorm(1-(1-c)/2)*s  # this computes the upper bound for the confidence interval
  list(lower=CI.l,upper=CI.u) # this tells the function what to print
}
```

For an example, suppose we take a random sample of 40 Canadian geese and have a sample mean of the weights of the cats to be 6 pounds with a standard error of 1.5 pounds, and we want a a 95% confidence interval for the average weight of a Canadian goose.

```
conf.int(6,.95,1.5)
```

```
## $lower
## [1] 3.060054
##
## $upper
## [1] 8.939946
```

Thus, the 95% confidence interval for the mean Canadian goose weight is [3.060054 lbs, 8.939946 lbs].

Standard Error for a Single Proportion

This is a function that calculates the standard error for a single proportion. Used in conjunction with the confidence interval function you only need to know the point estimate, number of observations, and confidence level and this will output the endpoints of the confidence interval.

```
#function to calculate the standard error for a single proportion
SE.p = function(p=proportion,n=sample.size) {
  SE=sqrt(p*(1-p)/n)
  return(SE)
}
# Example: single proportion confidence intervals
p=.3
n=100
s=SE.p(p,n);s
```

```
## [1] 0.04582576
```

```
c=.95
conf.int(p,c,s)
```

```
## $lower
## [1] 0.2101832
##
## $upper
## [1] 0.3898168
```

Standard Error for Difference of two Proportions

This is a function that calculates the standard error for the difference of two proportions. Used in conjunction with the confidence interval function you only need to know the point estimates, number of observations for each set, and confidence level and this will output the endpoints of the confidence interval.

```
#function to calculate the standard error for difference of two proportions
SE.pq = function(p=prop1,n=sample.size1,q=prop2,m=sample.size2) {
  SE=sqrt((p*(1-p)/n)+(q*(1-q)/m))
  return(SE)
}
#example
p=.5 #set first proportion
n=100 #number of observations for first set
q=.25 #set second proportion
m=50 #number of observations for second set
s=SE.pq(p,n, q,m);s
```

```
## [1] 0.07905694
```

```
# difference of two proportions confidence intervals
c=.95 #set confidence level
conf.int(p-q,c,s) #point estimate is p-q
```

```
## $lower
## [1] 0.09505124
##
## $upper
## [1] 0.4049488
```

```
# function to calculate pooled p hat
pooled.p = function(p=prop1,n=sample.size1,q=prop2,m=sample.size2) {
  phat=((p*n)+(q*m))/(n+m)
  return(phat)
}

# example: hypothesis test when H0:p=q
pp=pooled.p(p,n,q,m);pp
```

```
## [1] 0.4166667
```

```
#compute the test statistic
z=(p-q)/pp;z
```

```
## [1] 0.6
```

```
# compute the p-value for two-tailed test
2*(pnorm(-abs(z)))
```

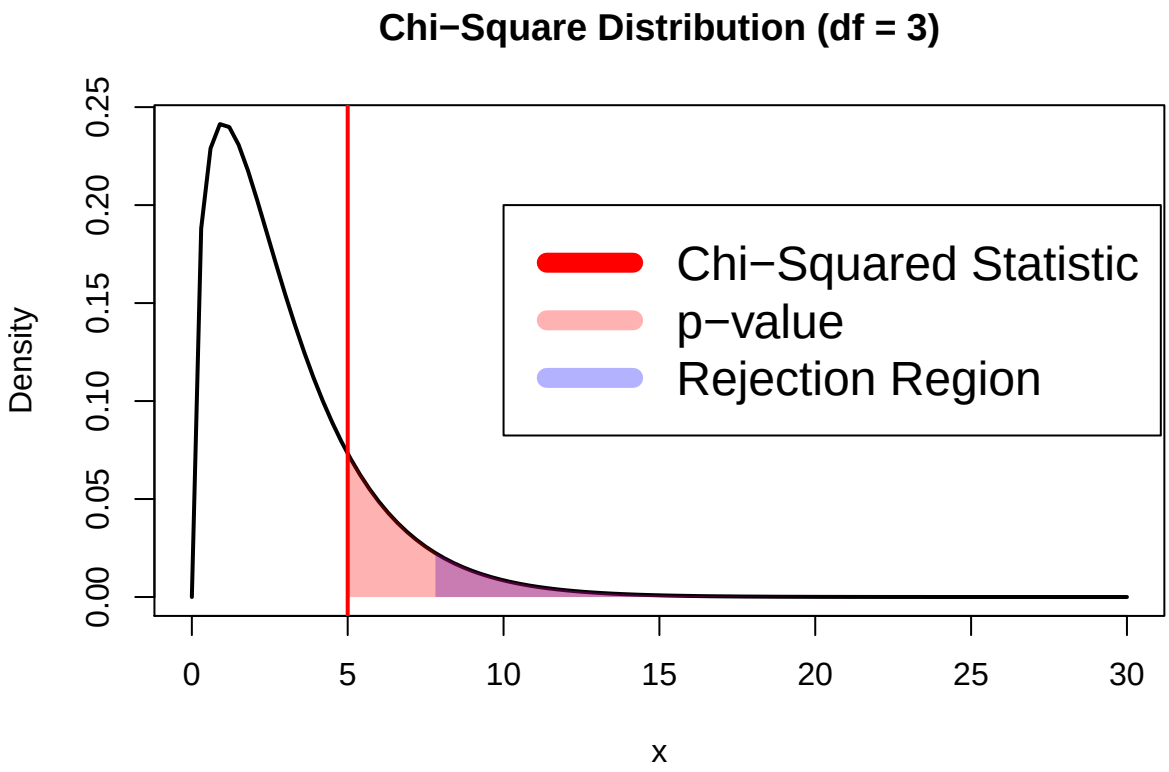
```
## [1] 0.5485062
```

Chi-squared Tests

```
# function that computes the chi-squared statistic parts
chi.s=function(o=observed,e=expected){
  cst=(o-e)^2/e
  return(cst)
}
#to find the chisquared statistic put all the pieces in a sum
c=sum(chi.s(o1,e1),chi.s(o2,e2),...,chi.s(ok,ek));c
#find the p-value with a chi-squared statistic
p=pchisq(c,3,lower.tail = FALSE);p
```

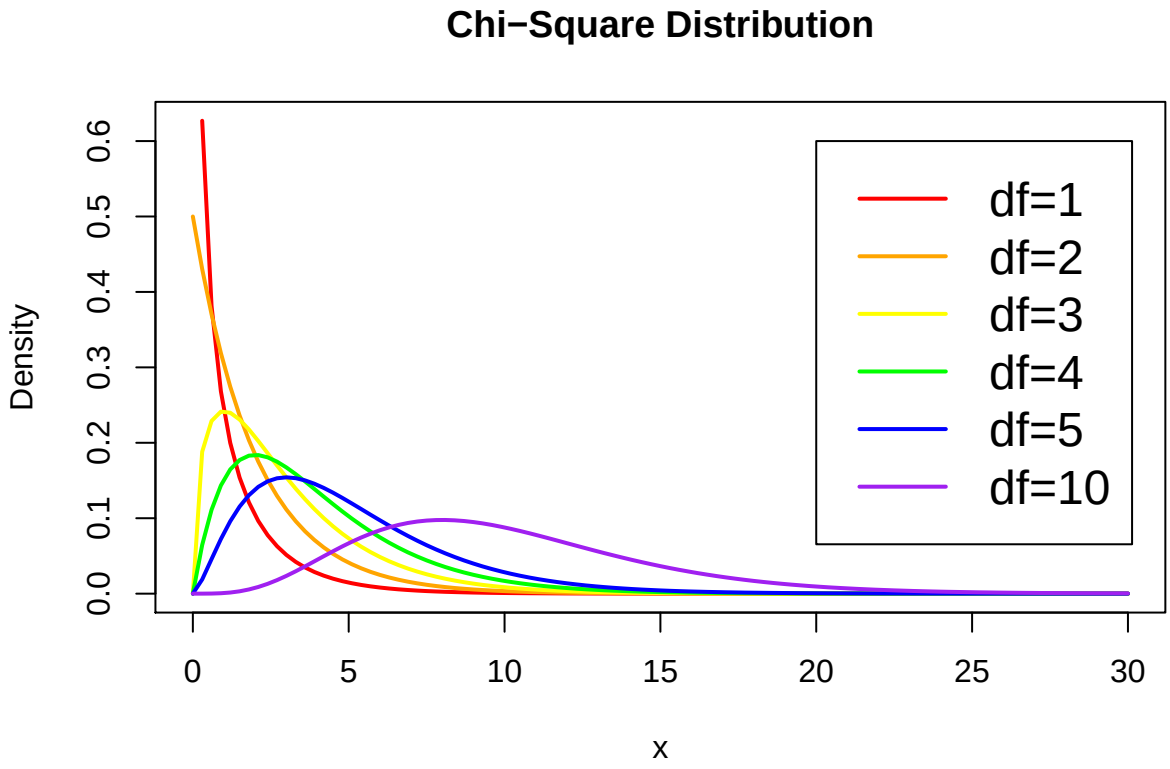
```
# chi-squared with p-value filled in and "rejection area" filled in
#these are the only two values you need to change:
d=3 # degrees of freedom
c=5 # chi-squared statistic

# create density curve
curve(dchisq(x, df = d), from = 0, to = 30,
      main = 'Chi-Square Distribution (df = 3)', # remember to change the title
      ylab = 'Density',
      lwd = 2)

#find upper values for 95% of distribution: this is for alpha=0.05
upper95 <- qchisq(.95, d)

#create vector of x values: this is for alpha=0.05
x_upper95 <- seq(upper95, 30)

#create vector of chi-square density values: this is for alpha=0.05
p_upper95 <- dchisq(x_upper95, df = d)

#fill in portion of the density plot for upper 95% value to end of plot
#this shows the rejection area
polygon(c(x_upper95, rev(x_upper95)), c(p_upper95, rep(0, length(p_upper95))),
        col = adjustcolor('blue', alpha=0.3), border = NA)

# set Chi-squared statistic
chisq <- c

#create vector of x values
x_chisq <- seq(chisq, 30)

#create vector of chi-square density values
p_chisq <- dchisq(x_chisq, df = d)

#fill in portion of the density plot for the p-value
polygon(c(x_chisq, rev(x_chisq)), c(p_chisq, rep(0, length(p_chisq))),
        col = adjustcolor('red', alpha=0.3), border = NA)
abline(v=c #c is the chisquared statistic
       ,lwd=2, col="red")
legend(10,.2,legend=c("Chi-Squared Statistic","p-value","Rejection Region"),
       col=c("red",adjustcolor('red', alpha=0.3),adjustcolor('blue', alpha=0.3)),
```

```
       lty=1, lwd=10,cex=1.5)
```



Plot of χ^2 distribution with different degrees of freedom.

```
#chi-squared with different degrees of freedom
curve(dchisq(x, df = 1), from = 0, to = 30,lwd=2,col="red",main = 'Chi-Square Distribution',ylab = 'Dens
curve(dchisq(x, df = 2),add=TRUE, from = 0, to = 30,lwd=2,col="orange")
curve(dchisq(x, df = 3),add=TRUE, from = 0, to = 30,lwd=2,col="yellow")
curve(dchisq(x, df = 4),add=TRUE, from = 0, to = 30,lwd=2,col="green")
curve(dchisq(x, df = 5),add=TRUE, from = 0, to = 30,lwd=2,col="blue")
curve(dchisq(x, df = 10),add=TRUE, from = 0, to = 30,lwd=2,col="purple")
legend(20,.6,legend=c("df=1", "df=2","df=3","df=4","df=5","df=10"),
       col=c("red", "orange","yellow", "green","blue", "purple"), lty=1, lwd=2,cex=1.5)
```



z tests

This code can be modified for any z test.

```
p.estimate=  #point estimate
expected= #expected
s= #sample standard deviation
n= #number of observations
SE=s/sqrt(n);SE #standarad error
z=(p.estimate-expected)/SE;z #zscore
p.val=pnorm(-abs(z));p.val #one tailed test
p.val=2*pnorm(-abs(z));p.val #two tailed test
```

This graphs the normal distribution, rejection area, p-value and z score.

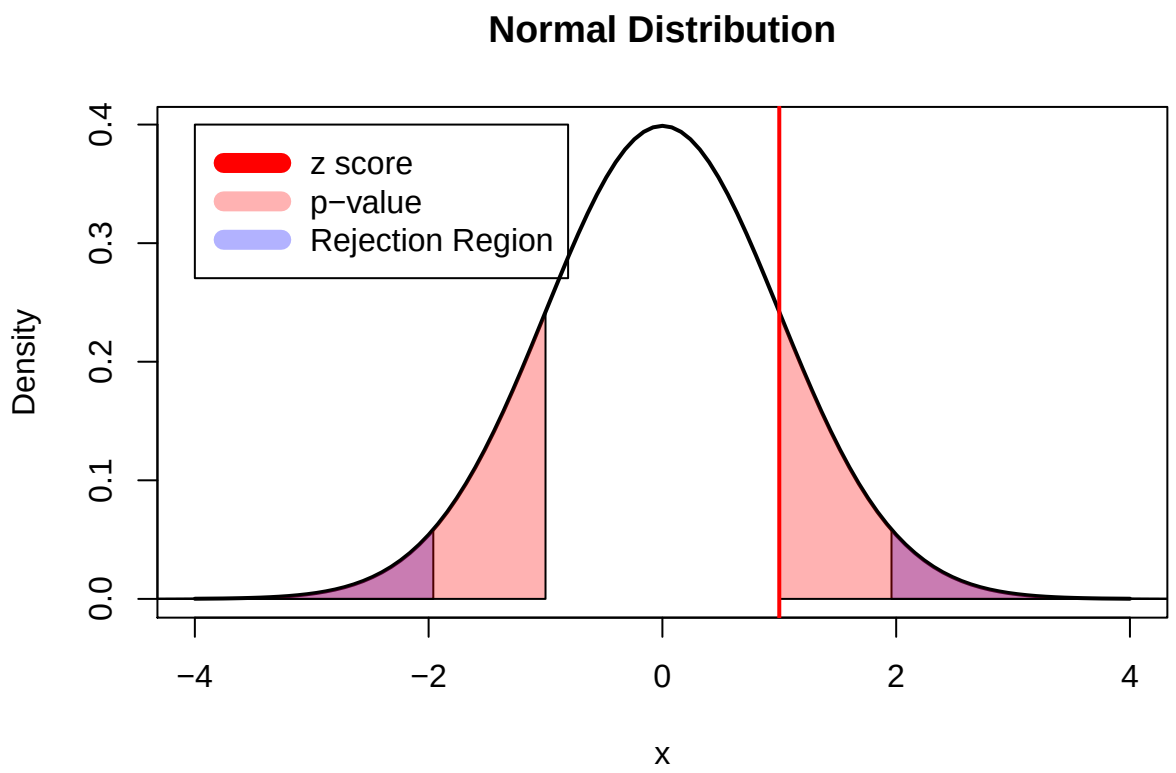
```
# Normal curve with p-value filled in and "rejection area" filled in
s = 1 # standard deviation
m = 0 # mean
a=.975 #significance level, check/change for one or two tailed tests
# create density curve
curve(dnorm(x, m, s), from = -4, to = 4,
      main = 'Normal Distribution', # remember to change the title
      ylab = 'Density',
      lwd = 2)
# The following two chunks are for filling in the rejection region
# Use only one for a one-tailed test
# Use both for a two-tailed test
# upper tail rejection region
lower.x = qnorm(a,m,s)
upper.x = 100
step = (upper.x - lower.x) / 10000
bounds = c(m-3*s, m+3*s)
cord.x = c(lower.x,seq(lower.x,upper.x,step),upper.x)
cord.y = c(0,dnorm(seq(lower.x,upper.x,step),m,s),0)
polygon(cord.x,cord.y,col=adjustcolor('blue', alpha=0.3))
#lower tail rejection region
lower.x1 = -100
upper.x1 = qnorm(1-a,m,s)
step1 = (upper.x1 - lower.x1) / 10000
bounds = c(m-3*s, m+3*s)
cord.x1 = c(lower.x1,seq(lower.x1,upper.x1,step1),upper.x1)
cord.y1 = c(0,dnorm(seq(lower.x1,upper.x1,step1),m,s),0)
polygon(cord.x1,cord.y1,col=adjustcolor('blue', alpha=0.3))
# The next two chunks fill in the p-value
# Only use one, the first chunk is for positive z score
# Second chunk is for negative z score
# p-value on right side
lower.z = 1 # positive z score
upper.z1 = 100
stepz = (upper.z1 - lower.z) / 10000
bounds = c(m-3*s, m+3*s)
cord.z1 = c(lower.z,seq(lower.z,upper.z1,stepz),upper.z1)
cord.z2 = c(0,dnorm(seq(lower.z,upper.z1,stepz),m,s),0)
```

```
polygon(cord.z1,cord.z2,col=adjustcolor('red', alpha=0.3))
#p-value on left side
lower.z1 = -100
upper.z = -1 #negative z score
step2 = (upper.z - lower.z1) / 10000
bounds = c(m-3*s, m+3*s)
cord.x2 = c(lower.z1,seq(lower.z1,upper.z,step2),upper.z)
cord.y2 = c(0,dnorm(seq(lower.z1,upper.z,step2),m,s),0)
polygon(cord.x2,cord.y2,col=adjustcolor('red', alpha=0.3))
# plot z score
abline(v=1 #z is the z-score
       ,lwd=2, col="red")
# add a legend
legend(-4,.4,legend=c("z score","p-value","Rejection Region"),
       col=c("red",adjustcolor('red', alpha=0.3),adjustcolor('blue', alpha=0.3)),
       lty=1, lwd=10,cex=1)
```

Example, for two tailed test at 0.05 significance level and z-score of 1

```
# Normal curve with p-value filled in and "rejection area" filled in
s = 1 # standard deviation
m = 0 # mean
a=.975 #significance level, check/change for one or two tailed tests
# create density curve
curve(dnorm(x, m, s), from = -4, to = 4,
      main = 'Normal Distribution', # remember to change the title
      ylab = 'Density',
      lwd = 2)
# The following two chunks are for filling in the rejection region
# Use only one for a one-tailed test
# Use both for a two-tailed test
# upper tail rejection region
lower.x = qnorm(a,m,s)
upper.x = 100
step = (upper.x - lower.x) / 10000
bounds = c(m-3*s, m+3*s)
cord.x = c(lower.x,seq(lower.x,upper.x,step),upper.x)
cord.y = c(0,dnorm(seq(lower.x,upper.x,step),m,s),0)
polygon(cord.x,cord.y,col=adjustcolor('blue', alpha=0.3))
#lower tail rejection region
lower.x1 = -100
upper.x1 = qnorm(1-a,m,s)
step1 = (upper.x1 - lower.x1) / 10000
bounds = c(m-3*s, m+3*s)
cord.x1 = c(lower.x1,seq(lower.x1,upper.x1,step1),upper.x1)
cord.y1 = c(0,dnorm(seq(lower.x1,upper.x1,step1),m,s),0)
polygon(cord.x1,cord.y1,col=adjustcolor('blue', alpha=0.3))

# The next two chunks fill in the p-value
# Only use one for one sided test, both for two sided
# first chunk is for positive z score
# Second chunk is for negative z score
# p-value on right side
```

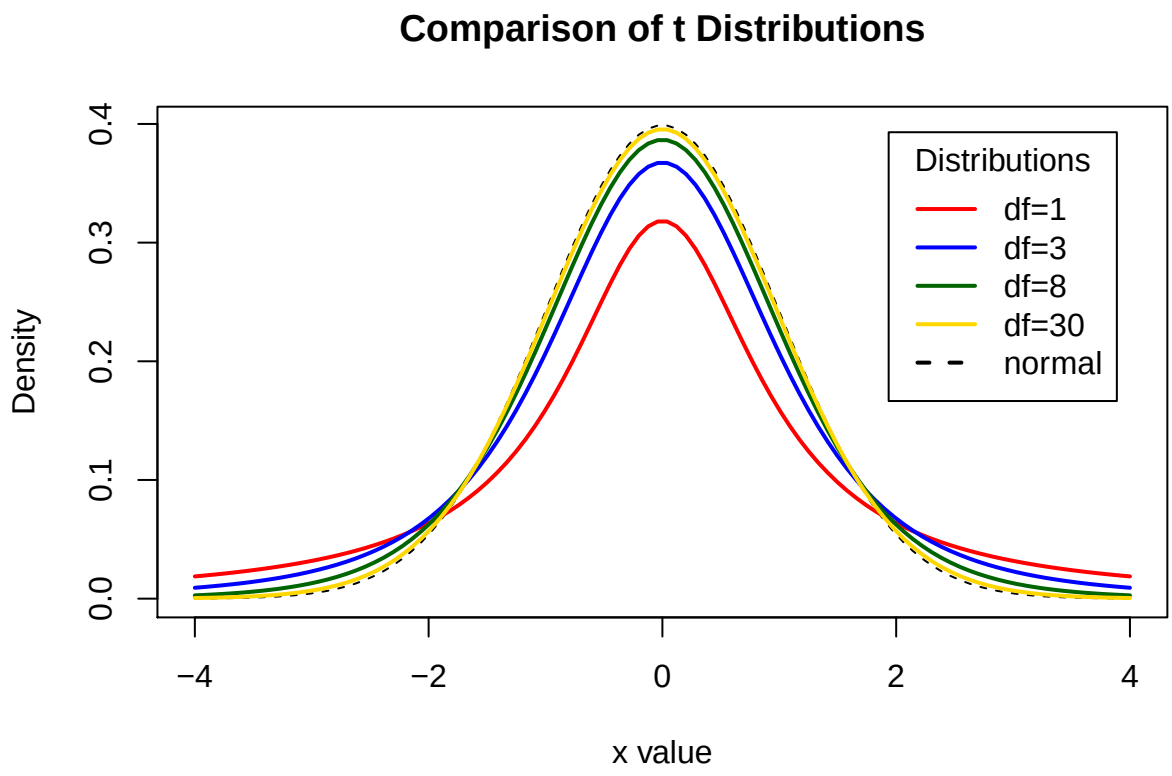
```r
lower.z = 1 # positive z score
upper.z1 = 100
stepz = (upper.z1 - lower.z) / 10000
bounds = c(m-3*s, m+3*s)
cord.z1 = c(lower.z,seq(lower.z,upper.z1,stepz),upper.z1)
cord.z2 = c(0,dnorm(seq(lower.z,upper.z1,stepz),m,s),0)
polygon(cord.z1,cord.z2,col=adjustcolor('red', alpha=0.3))

#p-value on left side
lower.z1 = -100
upper.z = -1
step2 = (upper.z - lower.z1) / 10000
bounds = c(m-3*s, m+3*s)
cord.x2 = c(lower.z1,seq(lower.z1,upper.z,step2),upper.z)
cord.y2 = c(0,dnorm(seq(lower.z1,upper.z,step2),m,s),0)
polygon(cord.x2,cord.y2,col=adjustcolor('red', alpha=0.3))

# plot z score
abline(v=1 #z is the z-score
       ,lwd=2, col="red")
# add a legend
legend(-4,.4,legend=c("z score","p-value","Rejection Region"),
       col=c("red",adjustcolor('red', alpha=0.3),adjustcolor('blue', alpha=0.3)),
       lty=1, lwd=10,cex=1)
```



t tests

This plots t distributions with different degrees of freedom as well as a normal curve for comparison.

```
x <- seq(-4, 4, length=100)
hx <- dnorm(x)

degf <- c(1, 3, 8, 30)
colors <- c("red", "blue", "darkgreen", "gold", "black")
labels <- c("df=1", "df=3", "df=8", "df=30", "normal")

plot(x, hx, type="l", lty=2, xlab="x value",
     ylab="Density", main="Comparison of t Distributions")

for (i in 1:4){
  lines(x, dt(x,degf[i]), lwd=2, col=colors[i])
}

legend("topright", inset=.05, title="Distributions",
       labels, lwd=2, lty=c(1, 1, 1, 1, 2), col=colors)
```



This code can be modified for any t-test

```
p.estimate= #this is the point estimate
expected= #the expected mean
s= #standard deviation of the data
n= #number of observations
SE=s/sqrt(n);SE #standard error for t-tests
t=(p.estimate-expected)/SE;t #find the T score
df=n-1 #degrees of freedom
p.val=pt(-abs(t), df);p.val #one tailed test
p.val=2*pt(-abs(t), df);p.val #two tailed test
```

This graphs the t distribution, rejection area, p-value and T score.

```
# t dist with p-value filled in and "rejection area" filled in
# Be aware of what you need for one and two tailed tests
a= #this sets the significance level
t= #set T-score
# create density curve
curve(dt(x,df), from = -4, to = 4,
      main = 't Distribution', # remember to change the title
      ylab = 'Density',
      lwd = 2)
#upper tail rejection area
lower.x = qt(a,df)
upper.x = 100
step = (upper.x - lower.x) / 10000
bounds = c(-4, 4)
cord.x = c(lower.x,seq(lower.x,upper.x,step),upper.x)
cord.y = c(0,dt(seq(lower.x,upper.x,step), df),0)
polygon(cord.x,cord.y,col=adjustcolor('blue', alpha=0.3))
#lower tail rejection area
lower.x1 = -100
upper.x1 = qt(1-a,df)
step1 = (upper.x1 - lower.x1) / 10000
bounds = c(-4, 4)
cord.x1 = c(lower.x1,seq(lower.x1,upper.x1,step1),upper.x1)
cord.y1 = c(0,dt(seq(lower.x1,upper.x1,step1),df),0)
polygon(cord.x1,cord.y1,col=adjustcolor('blue', alpha=0.3))
# p-value on right side
lower.t = t
upper.t1 = 100
stept = (upper.t1 - lower.t) / 10000
bounds = c(-4,4)
cord.t1 = c(lower.t,seq(lower.t,upper.t1,stept),upper.t1)
cord.t2 = c(0,dt(seq(lower.t,upper.t1,stept),df),0)
polygon(cord.t1,cord.t2,col=adjustcolor('red', alpha=0.3))
#p-value on left side
lower.t1 = -100
upper.t = -t
step2 = (upper.t - lower.t1) / 10000
bounds = c(-4,4)
cord.x2 = c(lower.t1,seq(lower.t1,upper.t,step2),upper.t)
cord.y2 = c(0,dt(seq(lower.t1,upper.t,step2),df),0)
```

```
polygon(cord.x2,cord.y2,col=adjustcolor('red', alpha=0.3))
#tscore
abline(v=-t #T score
       ,lwd=2, col="red")
legend(-4,.4,legend=c("T score","p-value","Rejection Area"),
       col=c("red",adjustcolor('red', alpha=0.3),adjustcolor('blue', alpha=0.3)),
       lty=1, lwd=10,cex=1)
```

Example L13.4

```
p.estimate=7.73
expected=8
s=0.77
n=25
SE=s/sqrt(n);SE
```

```
## [1] 0.154
```

```
t=(p.estimate-expected)/SE;t
```

```
## [1] -1.753247
```

```
df=n-1
p.val=pt(-abs(t), df);p.val #one tailed test
```

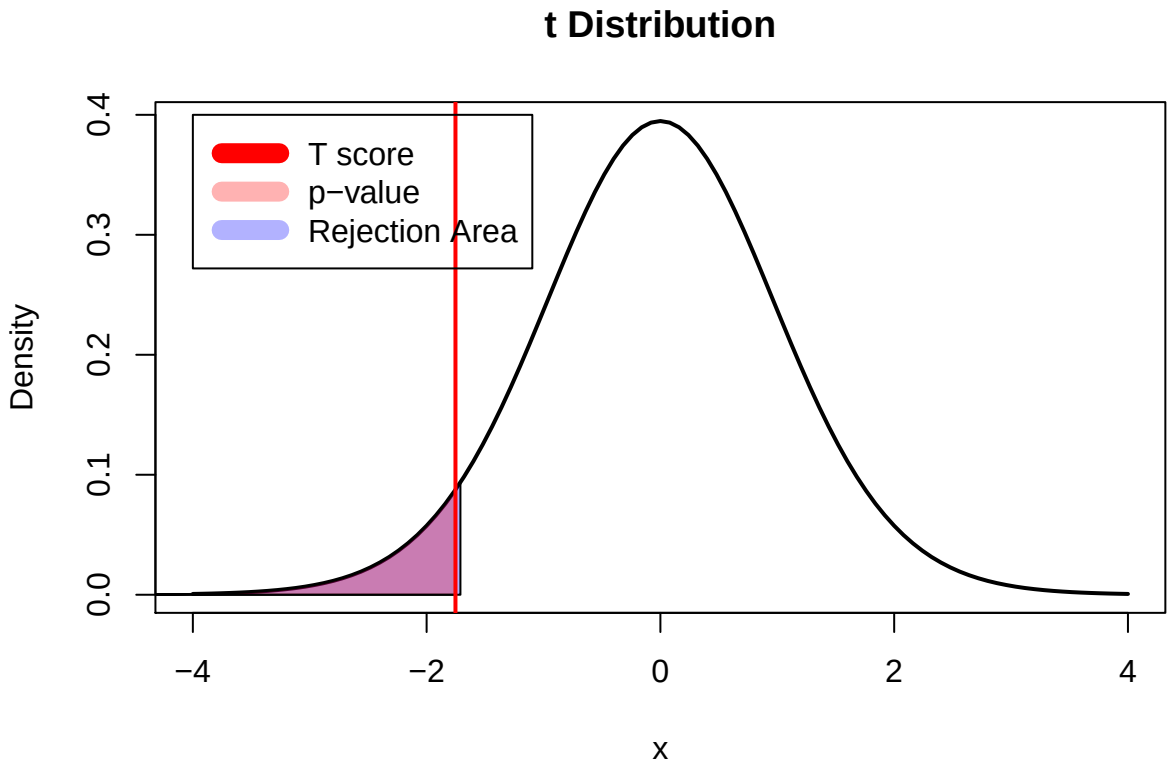
```
## [1] 0.04616261
```

```
p.val=2*pt(-abs(t), df);p.val #two tailed test
```

```
## [1] 0.09232523
```

```
# t dist with p-value filled in and "rejection area" filled in
a=.95 #this sets the significance level
# create density curve
curve(dt(x,df), from = -4, to = 4,
      main = 't Distribution', # remember to change the title
      ylab = 'Density',
      lwd = 2)
#lower tail rejection area
lower.x1 = -100
upper.x1 = qt(1-a,df)
step1 = (upper.x1 - lower.x1) / 10000
bounds = c(-4, 4)
cord.x1 = c(lower.x1,seq(lower.x1,upper.x1,step1),upper.x1)
cord.y1 = c(0,dt(seq(lower.x1,upper.x1,step1),df),0)
polygon(cord.x1,cord.y1,col=adjustcolor('blue', alpha=0.3))
#p-value on left side
lower.t1 = -100
upper.t = t
step2 = (upper.t - lower.t1) / 10000
```

```
bounds = c(-4,4)
cord.x2 = c(lower.t1,seq(lower.t1,upper.t,step2),upper.t)
cord.y2 = c(0,dt(seq(lower.t1,upper.t,step2),df),0)
polygon(cord.x2,cord.y2,col=adjustcolor('red', alpha=0.3))
#tscore
abline(v=t #T score
       ,lwd=2, col="red")
legend(-4,.4,legend=c("T score","p-value","Rejection Area"),
       col=c("red",adjustcolor('red', alpha=0.3),adjustcolor('blue', alpha=0.3)),
       lty=1, lwd=10,cex=1)
```



Paired Data Example L14.1

For more on this see the class slides or Example L14.1 document.

```
# Climate data from: https://www.openintro.org/data/index.php?data=climate70
climate=read.table("climate", header = TRUE)
#adds a column of the differences for dx70 to data frame
climate$dx70_diff<-climate$dx70_2018-climate$dx70_1948
#adds a column of the differences for dx90 to data frame
climate$dx90_diff<-climate$dx90_2018-climate$dx90_1948
str(climate)
```

```
## 'data.frame':    197 obs. of  9 variables:
##  $ station  : chr  "USC00203823" "USC00276818" "USC00186620" "USC00331890" ...
##  $ latitude : num  41.9 44.3 39.4 40.2 37.8 ...
##  $ longitude: num  -84.6 -71.3 -79.4 -81.9 -94.4 ...
##  $ dx70_1948: int  131 80 143 156 216 138 131 194 198 213 ...
##  $ dx70_2018: int  147 99 150 158 175 132 129 188 240 213 ...
##  $ dx90_1948: int  11 1 4 18 59 39 23 17 46 67 ...
##  $ dx90_2018: int  16 1 1 15 51 18 17 3 65 104 ...
##  $ dx70_diff: int  16 19 7 2 -41 -6 -2 -6 42 0 ...
##  $ dx90_diff: int  5 0 -3 -3 -8 -21 -6 -14 19 37 ...
```

```
head(climate)
```

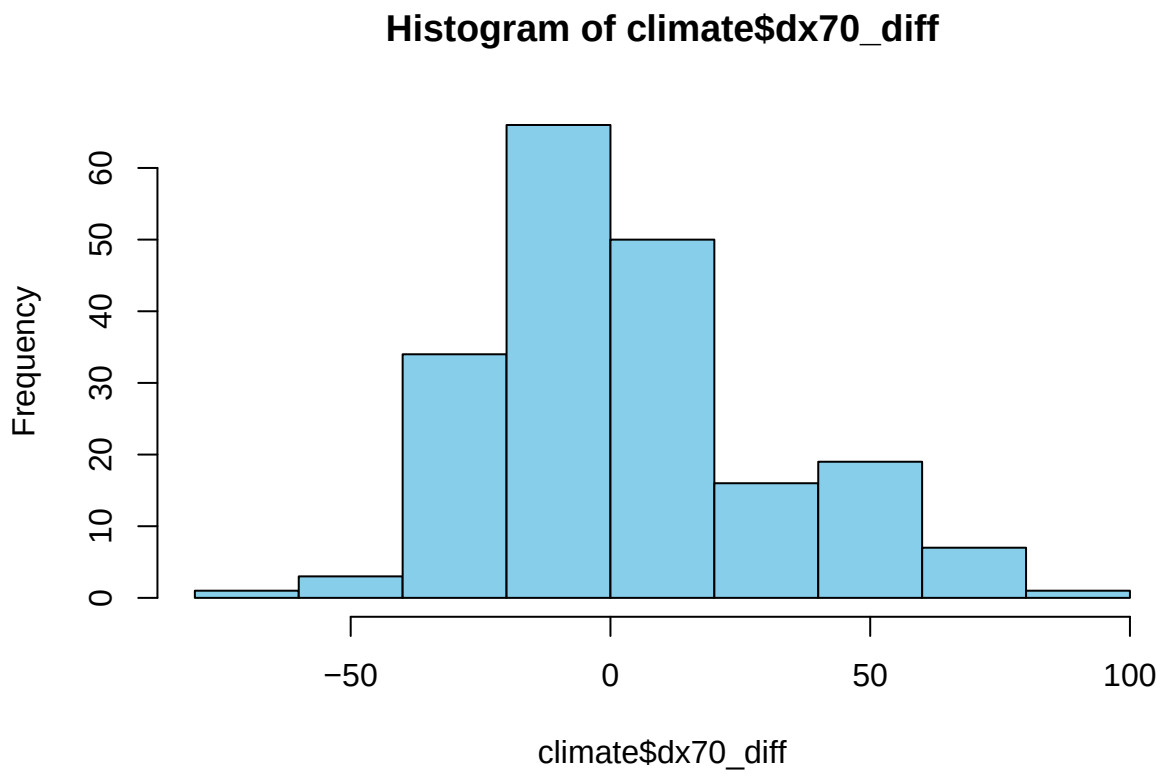
```
##       station latitude  longitude dx70_1948 dx70_2018 dx90_1948 dx90_2018
## 1 USC00203823 41.93520  -84.64110       131       147        11        16
## 2 USC00276818 44.25800  -71.25250        80        99         1         1
## 3 USC00186620 39.41317  -79.40025       143       150         4         1
## 4 USC00331890 40.24030  -81.87100       156       158        18        15
## 5 USC00235987 37.83950  -94.37400       216       175        59        51
## 6 USC00395691 45.56550 -100.44880       138       132        39        18
##   dx70_diff dx90_diff
## 1        16         5
## 2        19         0
## 3         7        -3
## 4         2        -3
## 5       -41        -8
## 6        -6       -21
```

```
tail(climate)
```

```
##         station latitude longitude dx70_1948 dx70_2018 dx90_1948 dx90_2018
## 192 GME00128014  49.8517    7.8725        80       158         1        18
## 193 USC00405187  35.4140  -86.8086       197       186        50        77
## 194 USC00137386  43.1791  -95.6602       141       134        19        12
## 195 USC00411048  30.1591  -96.3972       276       256       133       105
## 196 NLM00006235  52.9331    4.7500        19        71         0         1
## 197 USW00026411  64.8039 -147.8761        37        50         2         0
##     dx70_diff dx90_diff
## 192        78        17
## 193       -11        27
```

```
## 194        -7        -7
## 195       -20       -28
## 196        52         1
## 197        13        -2
```

```r
hist(climate$dx70_diff,col="skyblue")
```




```r
mean(climate$dx70_diff)
```

```
## [1] 4.086294
```

```r
sd(climate$dx70_diff)
```

```
## [1] 27.89846
```

```r
# z test
p.estimate=mean(climate$dx70_diff)
expected=0
s=sd(climate$dx70_diff)
n=197
SE=s/sqrt(n);SE
```

```
## [1] 1.987683
```

```r
z=(p.estimate-expected)/SE;z
```

```
## [1] 2.055808
```

```r
p.val=2*pnorm(-abs(z));p.val #two tailed test
```

```
## [1] 0.03980104
```

```r
# t test
p.estimate=mean(climate$dx70_diff)
expected=0
s=sd(climate$dx70_diff)
n=197
SE=s/sqrt(n);SE
```

```
## [1] 1.987683
```

```r
t=(p.estimate-expected)/SE;t
```

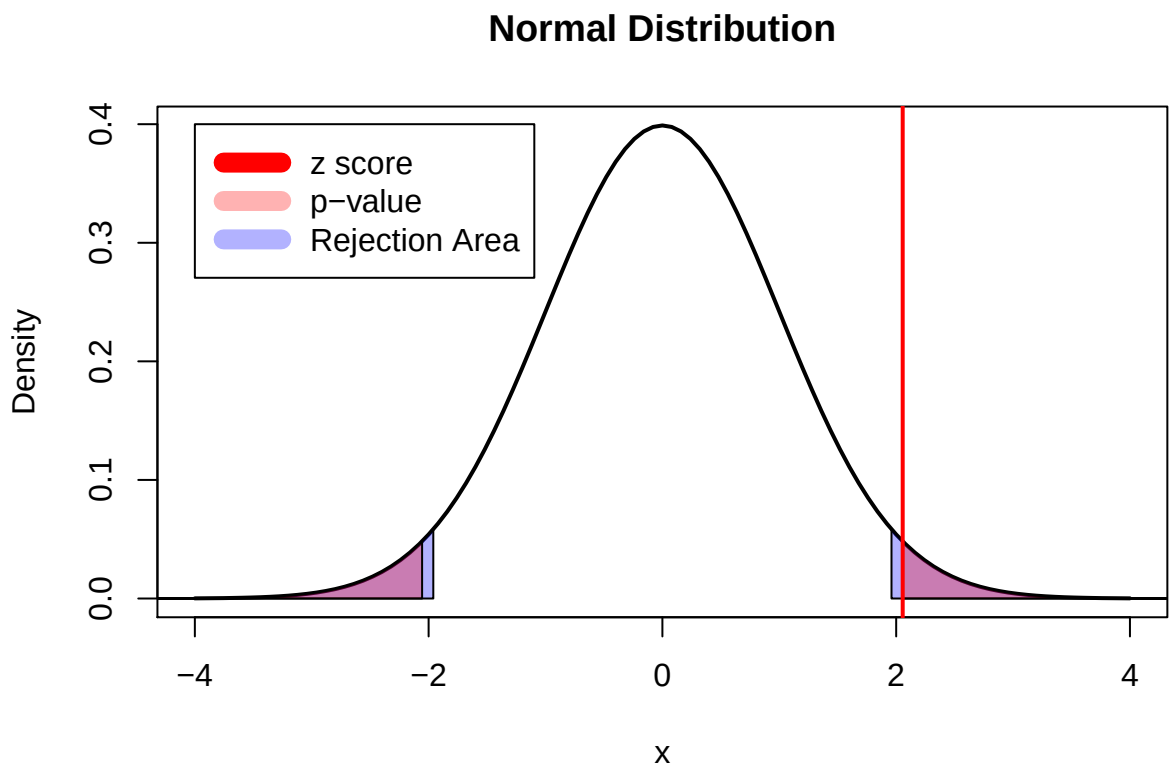
```
## [1] 2.055808
```

```r
df=n-1
p.val=2*pt(-abs(t), df);p.val #two tailed test
```

```
## [1] 0.04112662
```

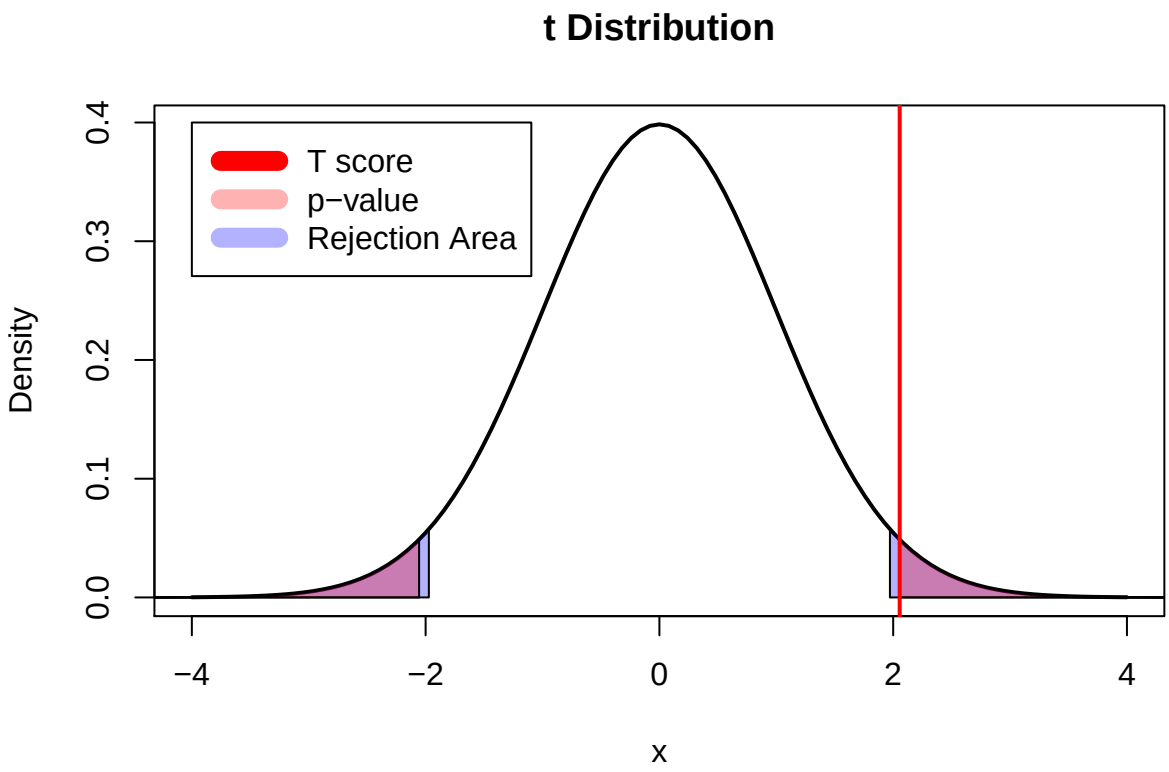
```r
#Plot for z
m=0;s=1;a=.975 # set parameters for standard normal
curve(dnorm(x, m,s), from = -4, to = 4,main = 'Normal Distribution',
          ylab = 'Density',
          lwd = 2)
#upper tail rejection area
lower.x = qnorm(a,m,s)
upper.x = 100
step = (upper.x - lower.x) / 10000
bounds = c(m-3*s, m+3*s)
cord.x = c(lower.x,seq(lower.x,upper.x,step),upper.x)
cord.y = c(0,dnorm(seq(lower.x,upper.x,step),m,s),0)
polygon(cord.x,cord.y,col=adjustcolor('blue', alpha=0.3))
#lower tail rejection area
lower.x1 = -100
upper.x1 = qnorm(1-a,m,s)
step1 = (upper.x1 - lower.x1) / 10000
bounds = c(m-3*s, m+3*s)
cord.x1 = c(lower.x1,seq(lower.x1,upper.x1,step1),upper.x1)
cord.y1 = c(0,dnorm(seq(lower.x1,upper.x1,step1),m,s),0)
polygon(cord.x1,cord.y1,col=adjustcolor('blue', alpha=0.3))
# p-value on right side
lower.z = abs(z)
upper.z1 = 100
```

```
stepz = (upper.z1 - lower.z) / 10000
bounds = c(m-3*s, m+3*s)
cord.z1 = c(lower.z,seq(lower.z,upper.z1,stepz),upper.z1)
cord.z2 = c(0,dnorm(seq(lower.z,upper.z1,stepz),m,s),0)
polygon(cord.z1,cord.z2,col=adjustcolor('red', alpha=0.3))
#p-value on left side
lower.z1 = -100
upper.z = -abs(z)
step2 = (upper.z - lower.z1) / 10000
bounds = c(m-3*s, m+3*s)
cord.x2 = c(lower.z1,seq(lower.z1,upper.z,step2),upper.z)
cord.y2 = c(0,dnorm(seq(lower.z1,upper.z,step2),m,s),0)
polygon(cord.x2,cord.y2,col=adjustcolor('red', alpha=0.3))
#zscore
abline(v=z ,lwd=2, col="red")
legend(-4,.4,legend=c("z score","p-value","Rejection Area"),
                col=c("red",adjustcolor('red', alpha=0.3),adjustcolor('blue', alpha=0.3)),
               lty=1, lwd=10,cex=1)
```




```
#plot for t
a=.975
#Draw density curve
curve(dt(x,df), from = -4, to = 4, main = 't Distribution', ylab = 'Density',lwd = 2)
#upper tail rejection area
lower.x = qt(a,df)
```

```
upper.x = 100
step = (upper.x - lower.x) / 10000
bounds = c(-4, 4)
cord.x = c(lower.x,seq(lower.x,upper.x,step),upper.x)
cord.y = c(0,dt(seq(lower.x,upper.x,step), df),0)
polygon(cord.x,cord.y,col=adjustcolor('blue', alpha=0.3))
#lower tail rejection area
lower.x1 = -100
upper.x1 = qt(1-a,df)
step1 = (upper.x1 - lower.x1) / 10000
bounds = c(-4, 4)
cord.x1 = c(lower.x1,seq(lower.x1,upper.x1,step1),upper.x1)
cord.y1 = c(0,dt(seq(lower.x1,upper.x1,step1),df),0)
polygon(cord.x1,cord.y1,col=adjustcolor('blue', alpha=0.3))
# p-value on right side
lower.t = abs(t)
upper.t1 = 100
stept = (upper.t1 - lower.t) / 10000
bounds = c(-4,4)
cord.t1 = c(lower.t,seq(lower.t,upper.t1,stept),upper.t1)
cord.t2 = c(0,dt(seq(lower.t,upper.t1,stept),df),0)
polygon(cord.t1,cord.t2,col=adjustcolor('red', alpha=0.3))
#p-value on left side
lower.t1 = -100
upper.t = -abs(t)
step2 = (upper.t - lower.t1) / 10000
bounds = c(-4,4)
cord.x2 = c(lower.t1,seq(lower.t1,upper.t,step2),upper.t)
cord.y2 = c(0,dt(seq(lower.t1,upper.t,step2),df),0)
polygon(cord.x2,cord.y2,col=adjustcolor('red', alpha=0.3))
#T score
abline(v=t ,lwd=2, col="red")
#legend
legend(-4,.4,legend=c("T score","p-value","Rejection Area"),
           col=c("red",adjustcolor('red', alpha=0.3),adjustcolor('blue', alpha=0.3)),
             lty=1, lwd=10,cex=1)
```


t Distribution
T score
p−value
Rejection Area
Density
0.0
0.1
0.2
0.3
0.4
−4
−2
0
2
4
x

Difference of two means

This finds the standard error for the difference of two means ($SE_{\overline{x_1}+\overline{x_2}}$).

```
SE.x.y=function(s1=standard.deviation1,n1=sample.size1,s2=standard.deviation2,n2=sample.size2){
  SExy=sqrt((s1^2/n1)+(s2^2)/n2)
    return(SExy)
}
```

This finds a confidence interval when using $t^{\star}_{df}$.

```
conf.int.t=function(p=point.estimate,c=confidence.level,s=standard.error, df=degrees.of.freedom)
{
  CI.l=p-qt(1-(1-c)/2,df)*s
  CI.u=p+qt(1-(1-c)/2, df)*s
  list(lower=CI.l,upper=CI.u)
}

#example L14.2
conf.int.t(3,.90,1.721918,9999)
```

```
## $lower
## [1] 0.1674345
##
## $upper
## [1] 5.832565
```

Linear Regression

```
# code for linear regression
# Remember to change all labels on plots
x= # independent data
y= # dependent data
cor(y,x) # gives correlation coefficient
plot(y~x, main="", xlab="", ylab = "") # scatter plot of data
model=lm(y~x) # creates linear regression model
summary(model) # gives coefficients/residuals/p-value
abline(model) # adds a line to the plot
plot(fitted(model),y,main="", xlab="", ylab = "") # plots the predicted vs actual
res=resid(model) #saves the residuals from the model
hist(res,main="", xlab="", ylab = "") # plots a histogram of the residuals
plot(res, fitted(model),main="", xlab="", ylab = "") # plots the residuals vs fitted values
```

Example for linear regression using the Iris data set.

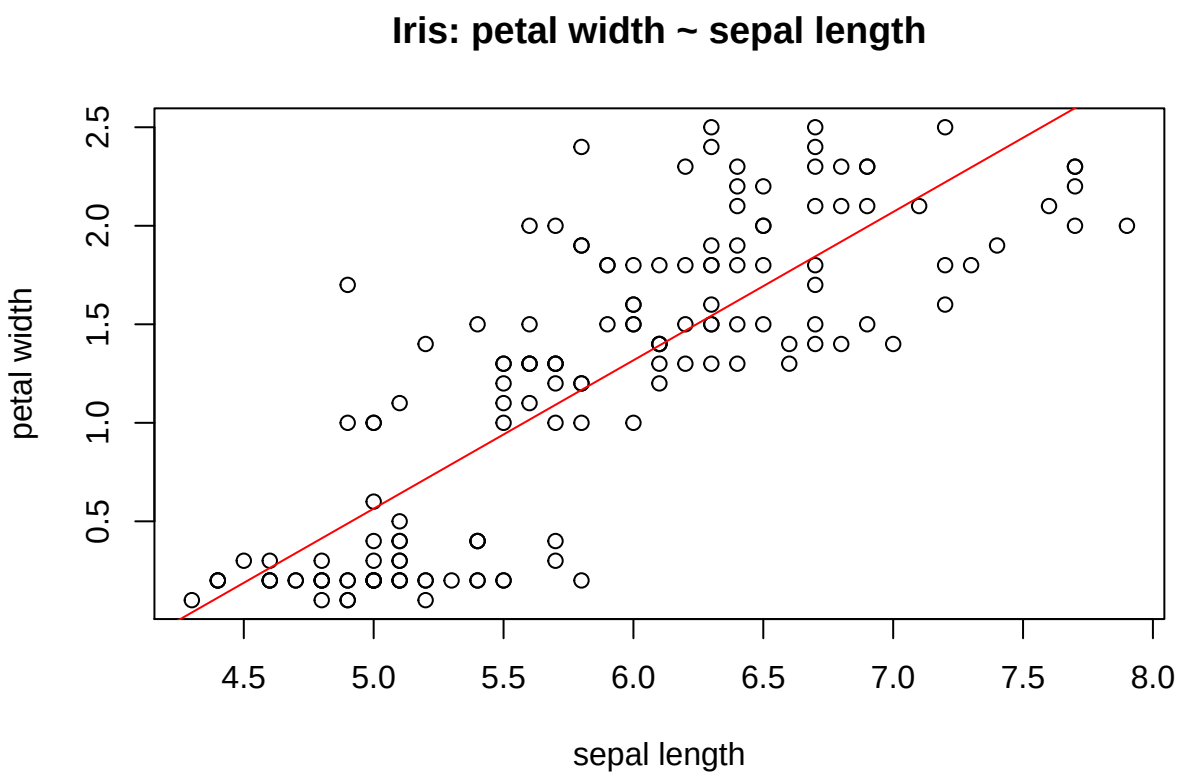
```
# example with iris data set
# independent data
x=iris$Sepal.Length
# dependent data
y=iris$Petal.Width
# gives correlation coefficient
cor(y,x)
```

```
## [1] 0.8179411
```

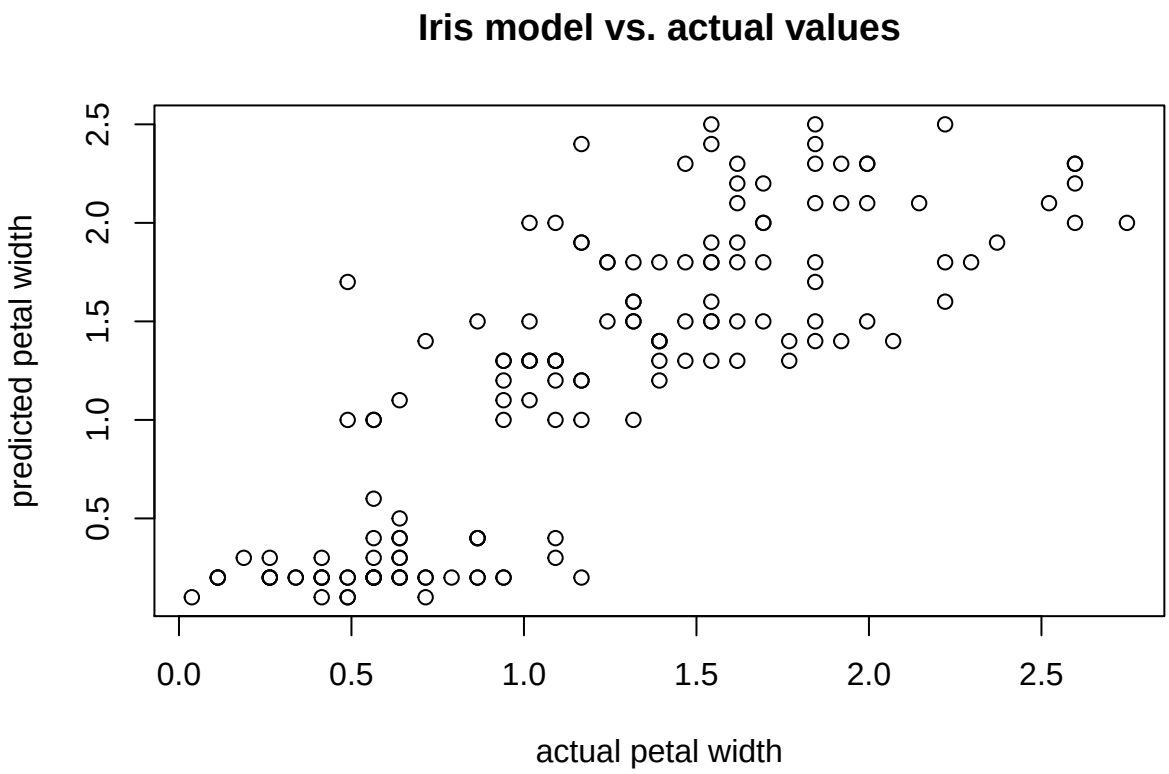
```
# scatter plot of data
plot(y~x, main="Iris: petal width ~ sepal length", xlab="sepal length", ylab = "petal width")
# creates linear regression model
model=lm(y~x)
# gives coefficients/residuals/p-value
summary(model)
```

```
##
## Call:
## lm(formula = y ~ x)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.96671 -0.35936 -0.01787  0.28388  1.23329
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -3.20022    0.25689  -12.46   <2e-16 ***
## x            0.75292    0.04353   17.30   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.44 on 148 degrees of freedom
## Multiple R-squared:  0.669,  Adjusted R-squared:  0.6668
## F-statistic: 299.2 on 1 and 148 DF,  p-value: < 2.2e-16
```

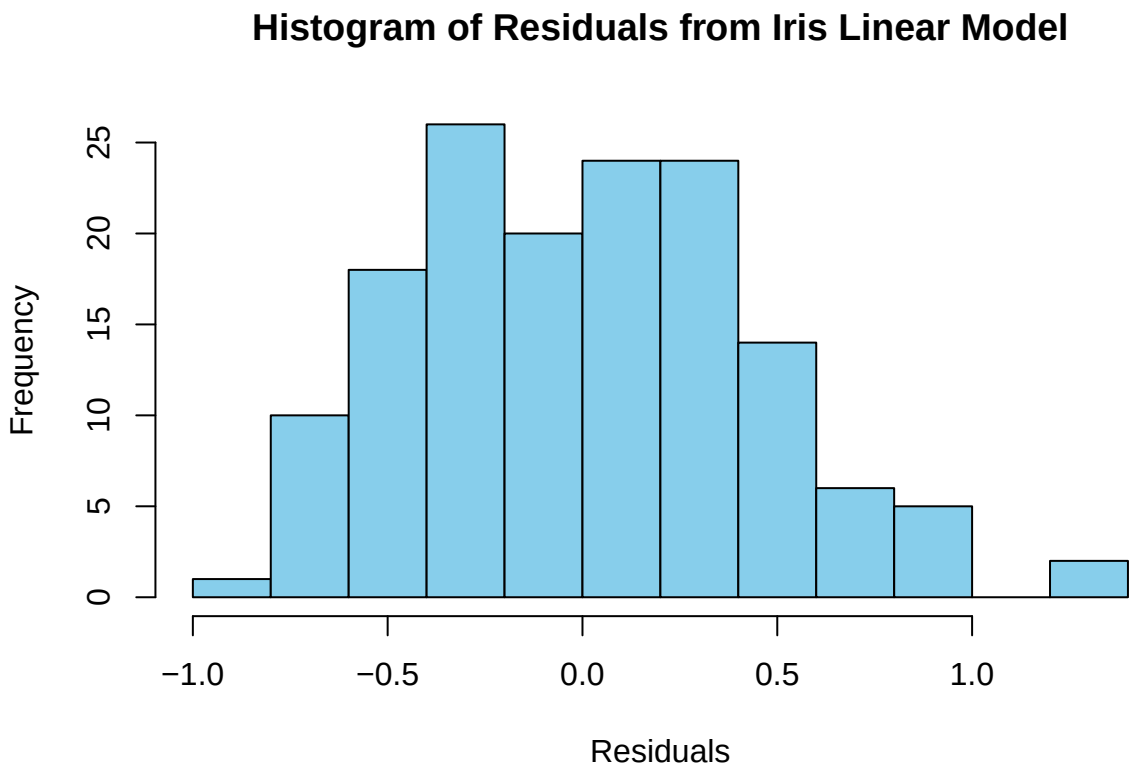
```r
# adds a line to the plot
abline(model, col="red")
```



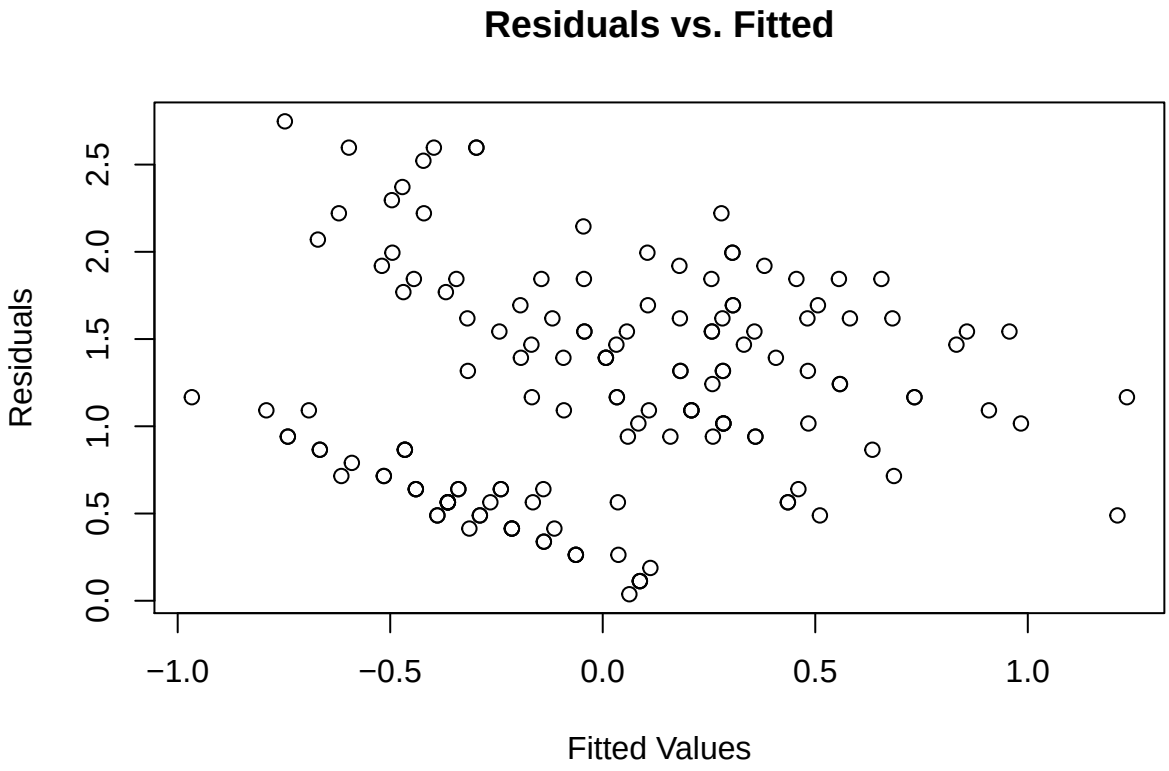

```r
# plots the predicted vs actual
plot(fitted(model),y,main="Iris model vs. actual values",
     xlab="actual petal width", ylab = "predicted petal width")
```




```r
#saves the residuals from the model
res=resid(model)
# plots a histogram of the residuals
hist(res,main="Histogram of Residuals from Iris Linear Model",
     xlab="Residuals", ylab = "Frequency",col="skyblue")
```




```
# plots the residuals vs fitted values
plot(res, fitted(model),main="Residuals vs. Fitted",
     xlab="Fitted Values", ylab = "Residuals")
```


Residuals vs. Fitted
Residuals
0.0
0.5
1.0
1.5
2.0
2.5
−1.0
−0.5
0.0
0.5
1.0
Fitted Values