Data Analysis

R Studio

Small Sample Test of Hypothesis and Confidence Interval About A Population Mean

The Long Way

```
x=c(0.5, 0.9, 4.5, 3.4, 1.0, 2.7, 1.1, 1.9, 0 ,0, 4.2, 2.1, 0, 2, 3.4, 3.4, 3.4, 2.5, 0.9, 5.1, 2.4)

mu0=1
xbar=mean(x)
stedv=sd(x)
n=length(x)
stderr=stdev/sqrt(n)
t=(xbar-mu0)/stderr
xbar
stdev
n
stderr
t
pvalue=pt(-abs(t), df=n-1)
pvalue

t=qt(0.975, df=n-1)
moe=t*stderr
lower=xbar-moe
upper=xbar+moe
t
lower
upper
```

The Short Way

Perform the t-test function…Alternative can be “two.sideded”, “less” or “greater”

```
#Load data

#Show all rows
head(BENZENE, 999)

tt=t.test (BENZENE$Benzene, alternative=c(“greater”), mu=1, conf.level=0.95)

names(tt)

tt$statistic

tt$p.value

tt=t.test(BENZENE$Benzene, alternative=c(“two.sided”), conf.level=0.95)
tt.conf.int

lower=tt$conf.int[1]
lower

upper=tt$conf.int[2]
upper
```

Small Sample Test of Hypothesis and Confidence Interval About A Paired Difference Mean

State the Hypothesis

H0: (mu1-mu2)=0 vs H1: (mu1-mu2)>0

The Long Way

```
new.method=c(77, 74, 82, 73, 87, 69, 66, 80)
std.method=c(72, 68, 76, 68, 84, 68, 61, 76)
diff=new.method-std,method
diff

mu0=0
xbar=mean(diff)
stdev=sd(diff)
n=length(diff)
stderr=stdev/sqrt(n)
t=(xbar-mu0)/stderr
xbar
stdev
n
stderr
t

pvalue=pt(-abs(t), df=n-1)
pvalue

t=qt(0.975, df=n-1)
moe=t*stderr
lower=xbar-moe
upper=xbar+moe
t
lower
upper
```

The Short Way

names() function swill show what is available

```
#Load the data

head(PAIREDSCORE, 999)

PAIREDSCORES$DIFF=PAIREDSCORES$NEW-PAIREDSCORES$STD

head(PAIREDSCORES, 999)

tt=t.test(PAIREDSCORES$DIFF, alternative=c("greater"), mu=0, conf.level=0.95)

names(tt)

tt$statistic

tt$p.value

tt=t.test(PAIREDSCORES$DIFF, alternative=c("two.sided"), conf.level=0.95)
tt$conf.int

#Show the lower and upper limits of the confidence interval
lower=tt$conf.int[1]
lower

upper=tt$conf.int[2]
upper
```

Small Sample Test of Hypothesis and Confidence Interval About a (Unpaired) Difference in Mean

State the hypothesis

H0: (mu1-mu2)=0 vs.H1: (mu1-mu2) not =0

The Long Way

```
new.method=c(80, 80, 79, 81, 76, 66, 71, 76, 70, 85)
std.method=c(79, 62, 70, 68, 73, 76, 86, 73, 72, 68, 75, 66)

#Descriptive statistics for the new method
xbar.new=mean(new.method)
stdev.new=sd(new.method)
n.new=length(new.method)
xbar.new
stdev.new
n.new

Descriptive statistics for the standard method
xbar.std=mean(std.method)
stdev.std=sd(std.method)
n.std=length(std.method)
xbar.std
stdev.std
n.std

#Find pooled standard deviation
pooled.variance=((n.new-1)*stdev.new^2+(n.std-1)*stdev.std^2)/(n.new+n.std-2)
pooled.stdev=sqrt(pooled.variance)
pooled.variance
pooled.stdev

#Find the observed value of the test statistic
mu0=0
xbar.diff=xbar.new-xbar.std
stderr.diff=sqrt(pooled.variance)*(1/n.new+1/n.std)
t=(xbar.diff-mu0)/stderr.diff
xbar.diff
stderr.diff
t
```

Use of –abs(t) turns left tail and right tail into a left tail. If it it a two-tailed test, I will multiply the value by two

```
#Find size of tail.
pvalue=2*pt(-abs(t), df=n.new+n.std-2)
pvalue

##As long as we're here, let's find the 95% confidence interval for the mean
t=qt(0.975, df=n.new+n.std-2)
moe.diff=t*stderr.diff
lower=xbar.diff-moe.diff
upper=xbar.diff+moe.diff
t
lower
upper
```

The Short Way

The paste() function will reveal it.

```
#Load the data
head(READING, 999)

#Split the data into new and std vectors
new=subset(READING, METHOD==“NEW     “)$SCORE
new

std=subset(READING, METHOD==“STD    “)$SCORE
std
```

This will also provide the confidence interval. This is NOT a paired difference test. When we use the pooled variance, we are assuming equal variances, so we will use var.equal=TRUE.

```
#Pereform two-sample t-test
tt=t.test(new, std, paired=FALSE, alternative=c("two.sided"). mu=0, conf.level=0.95, var.equal=TRUE)

#The names() function will show what is avaible
names(tt)

#Show the observed value of the test statistic and the p-value
tt$statistic
tt$p.value

#Find 95% confidence interval for the mean
tt$conf.int

#Show upper and lower limits of the confidence interval
lower=tt$conf.int[1]
lower

upper=tt$conf.int[2]
upper
```

Test Assumption of Equality of Variance

State the hypothesis:

H0: sigma1=sigma2 vs. sigma1 != sigma2

The Long Way

This is a quotient of variances and follows an F distribution. The numerator of the quotient is the higher variance.

```
new.method=c(80. 80, 79, 81, 76, 66, 71,76, 70, 85)
std.method=c(79, 62, 70, 68, 73, 76, 86, 73, 72, 68, 75, 66)

#Descriptive statistics for the new method
Var.new=var(new.method)
 var.new
n.new=length(new.method)
n.New

#Descriptive statistics for the standard method
var. std= var(std.method)
var.std
n.std=length(std.method)
n.std

#Find the observe value of the test statistic.
If (var.new > var.std){
    F=var.new/var.std
     numer.df=n.new-1
     denom.df=n.std-1
} else {
  F=var.std/var.new
  numer.df=n.std-1
  denom.df=n.new-1
}
F
numer.df
denum.df
```

```
#Find size of tail.
Pvalue=1-pf(F, df1=number.df, df2=denom.df)

pvalue=pvalue*2
pvalue
```

F statistic is the quotient of two squares, it is never negative. Multiply pvalue by two for a two-tail test

```
#Find Critical Value
cv=qf(.95, df1=numer.df, df2=denom.df)
Cv
```

The Short Way

```
#Load the data

#Show all rows
head(READING, 999)

#Use paste() function to reveal whitespace
head(paste(READING$METHOD), 999)

#Split the data into new and std vectors
new=subset(READING, METHOD==“NEW     “)$SCORE
new

std=subset(READING, METHOD==“STD     “)$SCORE
std
```

```
#Perform test
If (var(new)>var(std)){
    ftest=var.test(new, std)
} else {
  ftest=var.test(std, new)
)
Name(ftest)

#Extract the observed value of  the test statistic and the p-value
ftest$statistic

ftest$p.value
```

Simple Linear Regression

```
X=c(2,4,6,8)
Y=c(3,7,4,5,)

#Create a scatter plot
plot(x, y, main="Scatterplot", xlab="My X Variable", ylab="My Y Variable",
xlim=c(0,10), ylim=c(0, 10), lwd=2, cex=2, col="blue")

#Build the simple linear regression model
model=lm(y~x)
Summary(model)

#Display the betas, the r-squared, and the RMSE
summary=summary(model)
names(summary)

#Get the betas, r-squared, and rmse
beta0=summary$coefficients[1]; beta0
beta1=summary$coeffcients[2]; beta1
rsq=summary$r.squared
rsq
rmse=summary$r.squared
Rmse
```

Correlation

```
#Get the data

#Show the first six rows
head(CASINO)

#Plot data
plot(CASINO$EMPLOYEES, CASINO$CRIMERATE)

#Find the correlation using the cor() function
cor=cor(CASINO$EMPLOYEES, CASINO$CRIMERATE)

#Find the coefficient of determination (r squared) using the lm function
fit=lm(CRIMERATE~EMPLOYEES, data=CASINO)
summary=summary(fit)
rsq=summary$r.squared
Rsq

#For simple linear regression, correlation always has same sign as beta1
beta1=summary$coefficients[2]
beta1
r=sqrt(rsq)*sign(beta1)
r

#Find critical values
sigma=summary$sigma
sigma
ybar=mean(CASINO$CRIMERATE)
ybar
cv=sigma/ybar
cv
```

Confidence Interval and Prediction Interval

```
#Get the data

#Show the data
head(ADSALES, 999)

#Plot data
plot(ADSALES$ADVEXP_X, ADSALES$SALES_Y)

#Fit the regression model
model=lm(SALES_Y~ADVEXP_X, data=ADSALES)
Summary(model)

#New x values.
#Variable must have same name
to.predict=data.frame(ADVEXP_X=c(2,4))

#Point estimates
Predict(model, to.predict)

#Confidence intervals
Predict(model, to.predict, interval="confidence", level=.95)

#Prediction intervals
Predict(model, to.predict, interval="prediction", level=.95)
```

Simple Multiple Regression

Both two groups of plotting appear weak positive linear relationship. So we will get a quantitative measurement of those relationship--correlation

```
#Load the data

#Read the first six rows
head(GFLOCKS)

#Scatterplot
plot(GFCLOCK$NUMBIDS, GFCLOCKS$PRICE, lwd=2, cex=2, col="blue")
plot(GFCLOCKS$AGE, GFCLOCKS$PRICE, lwd=2, cex=2, col="blue")

#Correlation
cor(GFCLOCKS$NUMBIDS, GFCLOCKS$PRICE, method="pearson")
cor(GFCLOCKS$AGE, GFCLOCKS$PRICE, method="pearson")

#Find the least squares model
fit=lm(PRICE~AGE+NUMBIDS, data=GFCLOCKS)
summary=summary(fit)
summary

#Isolate the betas
names(summary)
summary$coefficients
beta0=summary$coefficients[1]
beta0

beta1=summary$coefficients[2]
beta1

beta2=summary$coefficients[3]
beta2
```

```
#Another way to isolate the betas
coeff=coefficients(fit)
coeff
beta0=coeff[1]
beta0
beta1=coeff[2]
beta1
beta2=coeff[3]
beta2

#Find the minimum value of Sum of Square Errors(SSE)
anova(fit)
```

Interaction Term

Adding an interaction term into the multiple regression model

```
#Load the data
head(GFCLOCKS)

#Fit the least squares model with interaction term
fit=lm(PRICE~AGE+NUMBIDS+AGE_BID, data=GFCLOCKS)
summary=summary(fit)
Summary

#Add the interaction term as a new column
GFCLOCKS$OUR_INTERACTION_TERM=GFCLOCKS$AGE*GFCLOCS$NUMBIDS
head(GFCLOCKS)

fit=lm(PRICE~AGE+NUMBIDS+OUR_INTERACTION+TERM, data=GFCLOCKS)
summary=summary(fit)
Summary

#Using an asterisk instead of a plus sign generates the interaction term automatically
fit=lm(PRICE~AGE*NUMBIDS, data=GFCLOCKS)
summary=summary(fit)
summary
```

Multiplicative Models

Make prediction:

Prediction is about salary specifically 12 years of experience, 16 years of education, female(0), 400 employees supervised and $160 million is assest

```
#Get the data
Head(EXECSAL)

#Create log, quadratic and interaction terms
EXECSAL$LNSAL=log(EXECSAL$SALARY)
EXECSAL$EXPSQ=EXECSAL$EXP^2
EXECSAL$GEN_SUP=EXECSAL$GENDER*EXECSAL$NONSUP
head(EXECSAL)

#Fit the model
fit=lm(LNSAL~EXP+EDUC+GENDER+NONSUP+ASSETS+EXPSQ+GEN_SUP,
data=EXECSAL)
summary(fit)

#Make prediction
PREDICT=data.frame(EXP=c(12), EDUC=c(16), GENDER=c(0), NONSUP=c(400),
ASSETS=c(160))
PREDICT$EXPSQ=PREDICT$EXP^2
PREDICT$GEN_SUP=PREDICT$GENDER*PREDICT$NONSUP
head(PREDICT)

#Make the prediction
LN_SALARY=predict(fit, PREDICT, interval="predict")
LN_SALARY

SALARY=exp(LN_SALARY)
SALARY
```

Quadratic Regression

```
#Get the data
head(TIRES)

#Scatterplot
Plot(TIRES$PRESS_X, TIRES$MILEAGE_Y, main="Plot of tire data", xlab="Pressure",
ylab="Mileage", xlim=c(29, 37), ylim=c(26, 40), lwd=2, cex=2, col="blue")

#Fit the model
fit=lm(MILEAGE_Y~PRESS_X, data=TIRES)
summary=summary(fit)
summary

#Fit the quadratic model
TIRES$PRESS_X_SQ=TIRES$PRESS_X^2
head(TIRES)
fit=lm(MILEAGE_Y~PRESS_X+PRESS_X_SQ, data=TIRES)
summary=summary(fit)
summary

#Predict the mileage for pressure of 31 and 34
PRESS_X=c(31,34)
PRESS_X_SQ=PRESS_X^2
PREDICT=data.frame(PRESS_X, PRESS_X_SQ)
head(PREDICT)

#Point estimate
predict(fit, PREDICT)

#Prediction interval
predict(fit, PREDICT, interval="predict")

#Confidence interval
predict(fit, PREDICT, interval="confidence")
```

Dummy (Qualitative) Variable

```
#Get the data
head(CARGO, 999)

#Use paste0 function to see the trailing blanks in the data
paste0(CARGO$CARGO)

#Create the dummy variables
CARGO$MYX1=ifelse (CARGO$CARGO ==”Fragile    “,1,0)
CARGO$MYX2=ifelse (CARGO$CARGO==”SemiFrag   “,1,0)
head(CARGO, 999)

#Fit the model
fit=lm(COST~MYX1+MYX2, data=CARGO)
summary(fit)

#R will treat categorical variables as dummy automatically
fit=lm(COST~CARGO, data=CARGO)
summary(fit)

#Use the levels parameter of the factor to indicate the order of the categories. The first listed will be treated as the reference level.
CARGO$FACTOR=factor(CARGO$CARGO, levels=c(“SemiFrag   “, “Durable    “, “Fragile    “))
fit=lm(COST~FACTOR, data=CARGO)
summary(fit)
```

Multiplicative Models

```
#Get the data
head(EXECSAL)

#Create log, quadratic, and interaction terms
EXECSAL$LNSAL=log(EXECSAL$SALARY)
EXECSAL$EXPSQ=EXECSAL$EXP^2
EXECSAL$GEN_SUP=EXECSAL$GENDER*EXECSAL$NONSUP
head(EXECSAL)

#Fit the model
fit=lm(LNSAL~EXP+EDUC+GENDER+NONSUP+ASSETS+EXPSQ+GEN_SUP,
data=EXESAL)
summary(fit)

#Create a data frame for prediction
PREDICT=data.frame(EXP=c(12), EDUC=c(16), GENDER=c(10),
ASSETS=c(160))
PREDICT$EXPSQ=PREDICT$EXP^2
PREDICT$GEN_SUP=PREDICT$GENDER*PREDICT$NONSUP
head(PREDICT)

#Make the prediction
LN_SALARY=predict(fit, PREDICT, interval="predict")
LN_SALARY


#Exponentiate the log of the salary to get salary
SALARY=exp(LN_SALARY)
SALARY
```

Quadratic Regression

```
#Read data from disk
head(TIRES)

#Scatterplot
plot(TIRES$PRESS_X, TIRES$MILEAGE_Y, main="Plot of tire data". xlab="Pressure", ylab="Mileage", xlim=c(29,37), ylim=c(26,40), lwd=2, cex=2, col="blue")

#Fit the model
fit=lm(MILEAGE_Y, PRESS_X, data=TIRES)
summary=summary(fit)
summary

#Fit the quadratic model
TIRES$PRESS_X_SQ=TIRES$PRESS_X^2
head(TIRES)

fit=lm(MILEAGE_Y~PRESS_X+PRESS_X_SQ, data=TIRES)
summary=summary(fit)
summary
```

```
#Predict the mileage of 31 and 34
PRESS_X=c(31,34)
PRESS_X_SQ=PRESS_X^2
PREDICT=data.frame(PRESS_X, PRESS_X_SQ)
head(PREDICT)

PREDICT=data.frame(PRESS_X=c(31,34))
PREDICT$PRESS_X_SQ=PREDICT$PRESS_X^2
head(PREDICT)

#Point estimates
predict(fit, PREDICT)

#Prediction interval
predict (fit, PREDICT, interval=”predict”)

#Confidence interval
predict(fit, PREDICT, interval=”confidence”)
```

Quantitative Independent Variables

```
#Get the data
head(MOSQUITTO, 999)

#Fit the model
MOSQUITTO$AveTempSQ=MOSQUITO$AveTemp^2
fit.before=lm(CatchRatio~AveTemp+AveTempSQ, data=MOSQUITO)
summary(fit.before)

#Calculate the correlation bewteen x and x =^2
cor.before=cor(MOSQUITO$AveTemp, MOSQUITO$AveTempSQ)
cor.before
```

```
#Form the equations
xbar=mean(MOSQUITO$AveTemp)
stdev=sd(MOSQUITO$AveTemp)
MOSQUITO$U=(MOSQUITO$AveTemp~xbar)/stdev
head(MOSQUITO, 999)

#Find U square
MOSQUITO$AveTempZSQ=MOSQUITo$AveTempZ^2
head(MOSQUITO, 999)

#Find the correlation between u and u^2
cor.after=cor(MOSQUITO$AveTempZ,
MOSQUITO$AveTempZSQ)
cor.after

#Fit the model
fit.after=lm(CatchRatio~AveTempZ+AveTempZSQ,
data=MOSQUITO
summary(fit.after)
```

Models with One Qualitative Independent Variable

```
#Get the data
head(BIDMAINT, 999)

#Use paste function to detect any blank
paste(BIDMAINT$STATE)

#Create dummy variable
BIDMAINT$KY=ifelse(BIDMAINT$STATE == “Kentucky”,1,0)
BIDMAINT$TX=ifelse(BIDMAINT$STATE==”Texa
s   “,1,0)
head(BIDMAINT, 999)

fit=lm(COST~KY+TX, data=BIDMAINT)
summary(fit)

#Find the interpret 95% confidence interval
confint(fit, “TX”, level=0.95)
```

Models with Two Qualitative Independent Variables

```
head(DIESEL, 999)

#Use paste function to detect blanks
paste(DIESEL$FUEL)

paste(DIESEL$BRAND)

#Create dummy variables
DIESEL$X1=ifelse(DIESEL$FUEL==”F2     “,1,0)
DIESEL$X2=ifelse(DIESEL$FUEL==”F3     “,1,0)
DIESEL$X3=ifelse (DIESEL$BRAND==”B2     “,1,0)
head(DIESEL, 999)

fit.without=lm(PERFORM~X1+X2+X3, data=DIESEL)
summary(fit.without)

#Fit the complete model for E(y)
DIESEL$x1x3=DIESEL$X1*DIESEL$X3
DIESEL$X2X3=DIESEL$X2*DIESEL$X3
head(DIESEL, 999)

fit.with=lm(PERFORM~X1+X2+X3+X1X3+X2X3, data=DIESEL)
summary(fit.with)
```

```
#Use the prediction equation for the model
fit.without$coefficients
beta0=fit.without$coefficients[1]
beta1=fit.without$coefficients[2]
beta2=fit.without$coefficients[3]
beta3=fit.without$coefficients[4]
beta0
beta1
beta2
beta3

x1=0, x2=1, x3=1
F3B2.without=beta)+X1*beta1+X2*beta2+X3*beta3
F3B2.without

fit.with$coefficients
```

```
beta0=fit.with$coefficients[1]
beta1=fit.with$coefficients[2]
beta2=fit.with$coefficients[3]
beta3=fit.with$coefficients[4]
beta4=fit.with$coefficients[5]
beta5=fit.with$coefficients[6]
beta0, beta1, beta2, beta3,
beta4, beta5
F3B2.with=beta0+x1*beta1+x2*
beta2+x3*beta3+x1*xbeta4+x2
*x3beta5
F3B2.with
```

Model Selection: Backward Elimination

#Starts will all variables

#Drops one variable at a time until dropping another variable no longer improves the model

#Once a variable is dropped it cannot re-enter the model

#WIth AIC criteria lower is better.

```
#Get the data
head(EXEXSAL2)


library(MASS)
full.model=lm(Y~X1+X2+X3+X4+X5+X6+X7+X8+X9+X10, data=EXEXSAL2)
model=step(full.model, direction="backward")
summary(model)
```

Model Selection: Forward Selection

#Adds one variable at a time until adding a new variable no long improves the model

#Once a variable is added, it never leaves the model

#With AIC criteria lower is better

```
min.model=lm(Y~1, data=EXEXSAL2)
#Intercept only model

biggest=formula(lm(Y~., data=EXEXSAL2))
#All variables including ID

model=step(min.model, direction='forward', scope=biggest)

summary(model)
```

Model Selection: Stepwise Regression

```
model=step(full.model, direction="both")

summary(model)
```

Model Selection:
Best Subset using Cp

#Using leaps library

#nbest shows n best models for each k predictors, for at most n*(k-1)+1 models

#For Mallow's Cp, want Cp "small and near" p (recall p=k+1)

#Can use method="Cp" then model$Cp, or method="adjr2" the nmodel$adjr2

```
library(leaps)

yvar=c("Y")

xvars=c("X1","X2","X3","X4","X5","X6","X7","X8","X9","X10")

model=leaps(x=EXEXSAL2[,xvars], y=EXEXSAL2[,yvar], names=xvars, nbest=3, method="Cp")

model$which

model$Cp
```

```
library(leaps)

yvar-c("Y")

xvars=c("X1","X2","X3","X4","X5","
X6","X7","X8","X9","X10")

model=leaps(x=EXEXSAL2[,xvars],y
=EXEXSAL2[,yvar], names=xvars,
nbest=3, method="adjr2")

model$which

model$adjr2
```

Demo create data partition (training/test data sets)

```
#Generate bogus dataframe
set.seed(1)
id=seq(from=1, to=20)
x=id+10
y=id+100
df=data.frame(id,x,y)
print(df)
```

```
#Create parittion index
library(caret)

#Splitting the data into 75%, 25% test
idx=createDataPartition(df$id, p=.75,
list=FALSE)
print(idx)

#Create and display the training
parition
train=df[idx,]
print(train)
```

```
#Create and display the testing partition
test=df[-idx,]

print(test)

#Here is another way to do it
sample=sample(1:nrow(df), 0.75*nrow(df))

sample

train=df[sample,]

train

test=df[-sample,]

test
```

Another way to split the data

```
sample=sample(1:nrow(df), 0.75*nrow(df))
sample

#75% for training
train=df[sample,]
train

#Remainder for testing
test=df[-sample]
test
```

Split the data into Training, Testing and Validation

data(iris)

idx=sample(seq(1,3), size=nrow(iris), replace=TRUE, prob=c(.8,.2,.2))

train=iris[idx==1,]

test=iris[idx==2,]

validation=iris[idx==3,]

```
1  data(iris)
2  idx=sample(seq(1,3), size=nrow(iris), replace=TRUE, prob = c(.8,.2,.2))
3  train=iris[idx==1,]
4  test=iris[idx==2,]
5  validation=iris[idx==3,]
6
7  dim(train)
8  dim(test)
9  dim(validation)
10
11 library(caret)
12 library(klaR)
13 library(MASS)
14
15 fit=NaiveBayes(iris$Species~., data=train)
16
17 fit=NaiveBayes(Species~., data=train)
18 p2=predict(fit, test[,1:4])
19
20 confusionMatrix(p2$class,test$Species)
```

Demo Cross-Validation

#Because model estimates are always blased to data they were built on

#This is a way to see how the model will work at predicting future values

#Also useful when you have to little data for training set and test set

```
#Generate independent variables
set.seed(1)
x1=round(runif(20, min=10, max=40))
x2=round(runif(20, min=10, max=40))
x3=round(runif(20, min=10, max=20))
e=round(rnorm(20, mean=0, sd=3))

#Set population parameters
beta0=-4
beta1=+1
beta2=-3
beta3=+1

y=beta0+beta1*x1+beta2*x2+beta3*x3+e

#Save it as a dataframe
df=data.frame(y, x1,x2,x3)
print(df)
```

```
#Identify candidate models first
#Save the RMSEs
sum1=summary(lm(y~x1+x2, data=df))
sum1
sigma1=sum1$sigma1
sigma1

sum2=summary(lm(y~x1+x3, data=df))
sum2
sigma2=sum2$sigma
sigma2

sum3=summary(lm(y~x2+x3, data=df))
sum3
sigma3=sum$sigma
sigma3

sum4=summary(lm(y~x1+x2+x3, data=df))
sum4
sigma4=sum4$sigma
sigma4
```

#The DAAG library is required for CVlm function
library(DAAG)

#Compare the sqrt of each ms with its corresponding RMSE
CVlm(data=df, m=5,
form.lm=formula(y~x1+x2), plotit=FALSE)
sigma1^2

#If you saw a significant difference bewteen the two then it could mean the model was overfitted to the training data

CVlm(data=df, m=5,
form.lm=formula(y~x1+x3), plotit=FALSE)
sigma2^2

#That is, it picked up the nuances of the training data rahter than the generalities.

CVlm(data=df, m=5,
form.lm=formula(y~x2+x3), plotit=FALSE)
sigma3^2

CVlm(data=df, m=5,
form.lm=formula(y~x1+x2+x3), plotit=FALSE)
sigma4^2

PRESS

```
#Get the data
head(EXPRESS)

#Fit the suggested model
EXPRESS$Wt_Dist=EXPRESS$Weight*EXPRESS$Distance
EXPRESS$Weight_SQ=EXPRESS$Weight^2
EXPRESS$Distance_SQ=EXPRESS$Distance^2
fit=lm(Cost~Weight+Distance+Wt_Dist+Weight_SQ+Distance_SQ,
data=EXPRESS)
summary(fit)

#Drop the Distance_SQ variable and Re-fit the model
fit2=lm(Cost~Weight+Distance+Wt_Dist+Weight_SQ,
data=EXPRESS)
summary(fit2)

#Find the PRESS statistic for each model
library("MPV")

#PRESS function is upper case
PRESS(fit)

press(fit2)
```

Detecting Multicollinearity in the Regression Model(VIF)

```
#Get the data
head(FTCCIGAR)

#Use cor function to detect multicollinearity
cor(FTCCIGAR)

#Fit full model
fit=lm(CO~TAR+NICOTINE+WEIGHT, data=FTCCIGAR)
summary(fit)

#Variance inflation factors
library(car)
vif(fit)
```

```
#VIF the long way
t_nw=lm(TAR~NICOTINE+WEIGHT,
data=FTCCIGAR)
r2=summary(t_nw($r.squared
vif_nic=1/(1-r2)
r2
vif_nic

w_nf=lm(WEIGHT~NICOTINE+TAR,
data=FTCCIGAR)
r2=summary)w_nt$r.squared
vif_wgt=1/(1-r2)
r2
vif_wgt
```

Reciprocal Transofmation of the Independent Variable

```
#Get the data
head(COFFEE,999)

#Plot the data
plot(COFFEE$PRICE, COFFEE$DEMAND, lwd=2, main="Before recprocal transform")


#Do the transform and plot again
COFFEE$PRICE_INV=1/COFFEE$PRICE
plot(COFFEE$PRICE_INV, COFFEE$DEMAND, lwd=2, main="After reciprocal transform")

#Fit the model
fit=lm(DEMAND~PRICE_INV, data=COFFEE)
summary(fit)
```

```
#Find a 95% confidence interval for the mean demand when the price is set at $32. per pound
to.predict=data.frame(PRICE=c(3.20))
to.predict$PRICE_INV=1/to.predict$PRICE
ci=predict(fit, to.predict, interval="confidence", level=.95)
ci

point.estimate=ci[1]
point.estimate

lower.limit=ci[2]
lower.limit

upper.limit=ci[3]
uppper.limit
```

Standardize Regression Coefficients

```
#Get the data
head(GFCLOCKS, 999)

#Fit the model without standardized coefficients
without=lm(PRICE~AGE+NUMBIDS, data=GFCLOCKS)
summary(without)

#Use Scale Function to standardize dependent and independent variables
GFCLOCKS$PRICE_Z=scale(GFCLOCKS$PRICE)
GFCLOCKS$AGE_Z=scale(GFCLOCKS$AGW)
GFCLOCKS$NUMBIDS_Z=scale(GFCLOCKS$NUMBIDS)
head(GFCLOCKS,999)

#Fit model with standardized coefficients
withstd=lm(PRICE_Z~AGE_Z+NUMBIDS_Z, data=GFCLOCKS)
summary(withstd)

#Can also use beta function
library(QuanPsyc)
lm.beta(without)--Compute standardize parameters
```

Transforms demos of transform of dependent variable

```
#These x values will be used throughout
x=seq(from=0, to=6, by=.5)
print(x)

#Reciprocal
y=1/(2+3*x)
plot(x,y)
lines(x,y)

y_inv=1/y
model1=lm(y_inv~x)
summary(model1)

#Square root
y=sqrt(2+3*x)
plot(x,y)
lines(x,y)

y_sq=y^2
model2=lm(y_sq~x)
summary(model2)
```

```
#Logarithmic
y=log(2+3*x)
plot(x,y)
lines(x,y)

exp_y=exp(y)
model3=lm(exp_y~x)
summary(model3)

#Exponential
y=exp(2+3*x)
plot(x,y)
lines(x,y)

ln_y=log(y)
model4=lm(ln_y~x)
summary(model4)
```

Detecting Unequal Variances

```
#Get the data
head(SOCWORK)

#Fit the model
SOCWORK$EXPSQ=SOCWORK$EXP^2
model=lm(SALARY~EXP+EXPSQ, data=SOCWORK)
summary(model)

#Find the y^'s and residuals
residuals=resid(model)
predictions=predict(model)

#Plot each y^ and its residual
plot(predictions, residuals, main="Residuals vs. Fitted Values", col="blue", lwd=2)
abline(h=0, col="red", lwd=2, lty=2)
```

Logarithm as a Stabilizing Transformation

```
#Get the data
head(SOCWORK)

#Fit the model
SOCWORK$LNSALARY=log(SOCWORK$SALARY)
SOCWORK$EXPSQ=SOCWORK$EXP^2
model=lm(LNSALARY~EXP+EXP+EXPSQ, data=SOCWORK)
summary=summary(model)
summary

#Find the y^'s and residuals
residuals=resid(model)
predictions=predict(model)

#Plot each y^ and its residual
plot(predictions, residuals, main="Residuals vs. Fitted Values)", col="blue",
lwd=2)
abline(h=0, col="red", lwd=2, lty=2)

#What about x^2 terms?
SOCWORK$LNSALARY=log(SOCWORJ$SALARY)
model=lm(LNSALARY~EXP, data=SOCWORK)
summary=summary(model)
summary

#Find the y^'s and residuals
residuals=resid(model)
predictions=predict(model)

#Plot each y^ and its residual
plot(predictions, residuals, main="residuals vs. Fitted Values", col="blue",
lwd=2)
abline(h=0, col="red", lty=2)
```

Other Diagnostics for Inflential Observations

```
#Get the data
head(FASTFOOD)

Mods to the data
FASTFOOD$BAD_SALES=FASTFOOD$SALES
FASTFOOD$BAD_SALES[13]=82


#Create Dummy Variables
FASTFOOD$X1=ifelse(FASTFOOD$CITY==1,1,0)
FASTFOOD$X2=ifelse(FASTFOOD$CITY==2,1,0)
FASTFOOD$X3=ifelse(FASTFOOD$CITY==3,1,0)

#Show the data
head(FASTFOOD, 20)

#Fit the model
model=lm(BAD_SALES~X1+X2+X3+TRAFFIC, data=FASTFOOD)
summary=summary(model)
summary

#A few influence diagnostics
library(MASS)
studentized.deleted.residuals=studres(model)
print(studentized.deleted.residuals)

plot(model)
```

Partial Residuals

```
#Get the data
head(OFFEE2)

#Part a
plot(COFFEE2$PRICE, COFFEE$DEMAND, main="Exploratory Data Analysis",
xlab="PRICE", ylab="DEMAND", lwd=2, col="red")

model=lm(DEMAND~PRICE+X, data=COFFEE2)
summary(model)

#Part b
residuals=resid(model)
plot(COFFEE2$PRICE, residuals, main="PRICE vs. RESIDUALS", xlab="PRICE",
ylab="RESIDUALS", col="blue", lwd=2)
abline(h=0, col="red", lwd=2, lty=2)

#Part c
library(car)
crPlots(model)

#Part d
COFFEE2$PRICE_INV=1/COFFEE2$PRICE
model=lm(DEMAND~PRICE_INV+X, data=COFFEE2)
summary(model)

#Plot the new residuals
residuals=resid(model)
plot(COFFEE2$PRICE, residuals, main="PRICE vs. RESIDUALS", xlab="PRICE",
ylab="RESIDUALS", col="blue", lwd=2)
abline(h=0, col="red", lwd=2, lty=2) #lty=2 creates dotted line
```

Matrix

```r
#Build a matrix from an array
M=matrix(c(2,1,-3,2), nrow=2, ncol=2, byrow=T)

#Build a matrix from a dataset
M=as.matrix(dataset)

#Add a substract two matrices
Sum=M + N, Diff= M - N

#Multiply two matrices
Prod = M %*% N

#Compute a matrix transpose
Trans=t(M)
Trans

#Convert a dataset into a matrix
M=as.matrix(ds)
```

Ridge Regression

Model is E(Y) = 0 + 1 X1 + 1 X2 + e with e~N(0,1)

Three variables are measured: x1,x2,x3.

x1 and x1 are U(0,1); x3=10 * X1 + unif(0,1).

This causes corr(X1,X3)=sqrt(100/101)=0.995.

We will fit OLS and ridge regressions to these data,

use the data to select the "best" constant to add,

and then evaluate the two regressions on a new test set.

runif(n, D): generates random compositions with a uniform distribution on the simplex. n, number of datasets to be simulated; D number of parts

```
# Ridge regression function, ridge.lm(), is on MASS package

library(MASS)

# Generating the data

set.seed(558562316)
N = 20      # Sample size

x1 = runif(N)
x2 = runif(N)
x3 = runif(N)
x3c = 10*x1 + x3 # New variable
ep = rnorm(N)
y = x1 + x2 + ep

ds = data.frame(y, x1, x2, x3, x3c)
plot(ds)

# OLS fit of 3-variable model using independent x3
ols = lm(y ~ x1 + x2 + x3)
summary(ols)

# OLS fit of 3-variable model using correlated x3.
olsc = lm(y ~ x1 + x2 + x3c)
summary(olsc)
```

lambda: Lambda is a measure of proportional reduction in error in cross tabulation analysis.

```
# Ridge regression using correlated variables
ridgec = lm.ridge (y ~ x1+x2+x3c, lambda = seq(0, .1, .001))
plot(ridgec)
select(ridgec)

# Selection of constant is at endpoint.  Extend endpoint and try again
ridgec = lm.ridge (y ~ x1+x2+x3c, lambda = seq(0, 1, .1))
plot(ridgec)
select(ridgec)

# Selection of constant is at endpoint.  Extend endpoint and try again
ridgec = lm.ridge (y ~ x1+x2+x3c, lambda = seq(0, 10, .01))
plot(ridgec)
select(ridgec)

# Final model uses lambda=4.
ridge.final = lm.ridge (y ~ x1+x2+x3c, lambda = 4)
ridge.final
```

```
# Compute the rse on the dataset.  Our regression has 4 parameters
# Including the intercept, so the degrees of freedom = #dataRows - 4 = 20-4 = 16
head(ds)
p.ols = predict(ols,newdata=ds[, c(2, 3, 4)])   # y ~ X1 + X2 + X3
p.olsc = predict(olsc,newdata=ds[, c(2, 3, 5)]) # y ~ X1 + X2 + X3c
p.ridge = coef(ridge.final)[1] + coef(ridge.final)[2]*ds$x1 +
  coef(ridge.final)[3]*ds$x2 + coef(ridge.final)[4]*ds$x3c

# Note that "coef(ridge.final)" lists all the coefficients
# including the intercept.  So this length is #var + 1
length(coef(ridge.final))

dof = length(y) - length(coef(ridge.final))
rse.ols = sqrt(sum((y - p.ols)^2) / dof)
rse.olsc = sqrt(sum((y - p.olsc)^2) / dof)
rse.ridge = sqrt(sum((y - p.ridge)^2) / dof)

rse.ols
rse.olsc
rse.ridge   # Doesn't seem to be doing much for us

plot(olsc, pch=16)
```

```
# build a predicted vs residual plot
resid.ridge = y - p.ridge
plot(p.ridge, resid.ridge, pch=16)

# Get the equivalent normal qq-plot
qqnorm(resid.ridge)
qqline(resid.ridge, col="red")

# Create test data and compute predicted values for OLS and ridge.
#  There's no predict() method for "ridgelm" objects
test = expand.grid(x1 = seq(.05,.95,.1), x2 = seq(.05,.95,.1), x3=seq(.05,.95,.1))
mu = test$x1 + test$x2
test$x3c = 10*test$x1 + test$x3
pred.ols = predict(ols,newdata=test)   # y ~ X1 + X2 + X3
pred.olsc = predict(olsc,newdata=test) # y ~ X1 + X2 + X3c
pred.ridge = coef(ridge.final)[1] + coef(ridge.final)[2]*test[,1] +
  coef(ridge.final)[3]*test[,2] + coef(ridge.final)[4]*test[,4]

coef(ridge.final)

# Compute the mean square prediction error
pdof = length(mu) - length(coef(ridge.final))
RSPE.ols = sqrt(sum((pred.ols - mu)^2)/dof)
RSPE.olsc = sqrt(sum((pred.olsc - mu)^2)/dof)
RSPE.ridge = sqrt(sum((pred.ridge - mu)^2)/dof)

# Print them out
RSPE.ols
RSPE.olsc
RSPE.ridge  # A bit better here
```

```
# Notice that we do not get much in the way of diagnostics out
# of lm.ridge ... sigh

# There is another package called "penalized" which implements both lasso (L1) and ridge (L2)
# regression, but its computation is more complicated as an optimization
# technique and its output is not comparable.
#
# I provide this initial example to get you started if you want to explore
# this package in more depth.
library(penalized)
pfit = penalized(y ~ x1 + x2 + x3c, lambda2 = 4)
summary(pfit)
print(pfit)
coefficients(pfit, "all")
penalty(pfit)
head(residuals(pfit))

plot(fitted(pfit), residuals(pfit))
```

Lasso

```
library(glmnet)
library(corrplot)
library(MASS)
load("QuickStartExample.RData")

# This dataset has an x-variable set with twenty columns
# and a y-variable set with one column.
#
# The datasets here need to be matrices
head(x)
head(y)

s = sample(1:100, 60)
xTrain = x[s, ]
yTrain = y[s, ]
xTest = x[-s, ]
yTest = y[-s, ]

# Let's run a quick regression on this
ds = data.frame(yTrain, xTrain)
names(ds)[1] = "Y"
head(ds)
fitOld = lm(Y ~ ., data=ds)
summary(fitOld)
yHat = predict(fitOld, data.frame(xTest))
c = fitOld$coefficients
dof = length(yTest) - length(c)
rsePredict = sqrt(sum((yTest - yHat)^2) / dof)
rsePredict
```

```
# The glmnet function
# Defaults to lasso, but can also do a version of ridge
fit = glmnet(xTrain, yTrain)  # note there is no ~ here and y comes after the x's.
plot(fit, label=T)

summary(fit)  # not much help :)
print(fit)
names(fit)

# Let's look at the coefficients for the model at a specific lambda
# Note: It selected all of the variables with some contribution
coef(fit, s=.004416)    # Why s and not lambda ... sigh

c = as.matrix(coef(fit, s=.004416))
yHat = predict(fit, xTrain, s=.003154)
dof = length(yTrain) - length(c)
rse = sqrt(sum((yTrain - yHat)^2) / dof)  # Still 20 variables in the regression
rse
head(yHat)
head(yTrain)

yPredict = predict(fit, xTest, s=.003154)
dof = length(yTest) - length(c)
rsePredict = sqrt(sum((yTest - yPredict)^2) / dof)
rsePredict

head(yHat)
head(yTrain)
```

```
# One of the nice things about this function is its
# Ability to test with cross validation to choose the lamba

cvfit = cv.glmnet(xTrain, yTrain)
par(mar=c(3, 3, 3, 3))
plot(cvfit)

print(cvfit$lambda.min)
coef(cvfit, s="lambda.min")  # Note we are getting a lot more contributions here

yPredict = predict(cvfit, newx=xTest, s="lambda.min")
dof = length(yTest) - length(c)
rsePredict = sqrt(sum((yTest - yPredict)^2) / dof)
rsePredict
```

Multicolinearity

EXAMPLE 1

rnorm(n, mean, sd):

```
library(car)
library(MASS)
library(clusterGeneration)
library(corrplot)

# The dataset "colinearData.csv" was generated with the following code
x1 = rnorm(100)
x2 = -x1 + rnorm(100, 0, .3)
y = 2.5 - .05 * x1 + rnorm(100, 0, .2)

d = data.frame(y, x1, x2)
# write.table(d, "colinearData.csv", sep=",", row.names=F)

# Let's read it in and fit a complete model
d = read.table("colinearData.csv", sep=",", header=T)
head(d)
plot(d)

fit = lm(y ~ x2 + x1, data=d)
summary(fit)

fit = lm(y ~ x1, data=d)
summary(fit)
```

EXAMPLE 2

```
Y = c(5, 3, 9, 9, 13, 11, 17, 15)
X1 = c(6, 8, 8, 10, 10, 12, 12, 14)
X2 = c(13, 13, 11, 11, 9, 9, 7, 7)

d = data.frame(Y, X1, X2)
cor(d)
plot(d)

# Note the coefficeints we get here!
summary(lm(Y ~ X1, data=d))
summary(lm(Y ~ X2, data=d))

# Now, when we include both ... beta_1 changes sign!
summary(lm(Y ~ X1 + X2, data=d))
vif(lm(Y ~ X1 + X2, data=d))
```

EXAMPLE 3

```
Y = c(3.7, 3.7, 4.2, 4.3, 5.1, 5.2, 5.2, 5.6, 5.6, 6.0)
X1 = c(3.2, 3.3, 3.7, 3.3, 4.1, 3.8, 2.8, 2.6, 3.6, 4.1)
X2 = c(2.9, 4.2, 4.9, 5.1, 5.5, 6.0, 4.9, 4.3, 5.4, 5.5)

d = data.frame(Y, X1, X2)
cor(d)
plot(d)

# Note the coefficeints we get here!
summary(lm(Y ~ X1), data=d)
summary(lm(Y ~ X2), data=d)

# Now, when we include both ... beta_1 changes sign!
summary(lm(Y ~ X1 + X2), data=d)
vif(lm(Y ~ X1 + X2), data=d)

# Note:  VIF scores here are low ... the parameter changes sign because of its
# insignificance!
```

EXAMPLE 4

nVars: number of variables

nObs: Compute the number of non-missing observations. Provides a 'default' method to handle vectors, and a method for data frames

```
# The first part of this shows how to build a random dataset with a specific covariance
# profile.  If you wish to skip the technical details, proceed below to the regression
# analysis of this synthetic set
#
# Now, let's look at a more complicated situation.  We build a multivariate dataset
# With a known (random) covariance matrix

set.seed(2)  # This makes the pseudorandom generator give us the same data every time
nVars = 15
nObs = 200
covMat = genPositiveDefMat(nVars,covMethod="unifcorrmat")$Sigma
xVars = mvrnorm(nObs,rep(0,nVars),Sigma=covMat)

corMat = cor(xVars)
corMat

corrplot(corMat, method = "ellipse")
head(xVars)

parms = runif(nVars,-10,10)   # Set of uniform random parameters
y = xVars %*% matrix(parms) + rnorm(nObs,sd=20)

dataset = data.frame(y, xVars)
write.table(dataset, file="multicolinearData.csv", sep=",", row.names=F, col.names=T)

# A quick way to get the list of parameter names for the regression
# Otherwise we would have to manually type X1 + X2 + ...
# The -1 eliminates the first parameter, which is our dependent variable "y"
regFormula = paste('y ~', paste(names(dataset)[-1], collapse=' + '))
fit = lm(regFormula, data=dataset)
```

paste(): Concatenate vectors after converting to character

```
# Same as
fit = lm(y ~ X1 + X2 + X3 + X4 + X5 + X6 + X7 + X8 + X9 + X10 + X11 + X12 + X13 + X14 + X15, data=dataset)
summary(fit)
parms

vif(fit)

# Look for the parameter with the biggest value here and then remove that parameter in the
# next regression formula.  Notice that since "y" is the first column, X8 will be the 9th
# parameter, for example, so you will always remove "n+1"
regFormula = paste('y ~ ', paste(names(dataset)[c(-1,-9)], collapse=' + '))
fit = lm(regFormula, data=dataset)

# Same as
fit = lm(y ~ X1 + X2 + X3 + X4 + X5 + X6 + X7 + X9 + X10 + X11 + X12 + X13 + X14 + X15, data=dataset)
summary(fit)
parms

vif(fit)

regFormula = paste('y ~', paste(names(dataset)[c(-1,-8,-9)], collapse=' + '))
fit = lm(regFormula, data=dataset)

# Same as
fit = lm(y ~ X1 + X2 + X3 + X4 + X5 + X6 + X9 + X10 + X11 + X12 + X13 + X14 + X15, data=dataset)
summary(fit)
parms

vif(fit)

# Then remove
regFormula = paste('y ~', paste(names(dataset)[c(-1,-8,-9,-5)], collapse=' + '))
fit = lm(regFormula, data=dataset)

# Same as
fit = lm(y ~ X1 + X2 + X3 + X5 + X6 + X9 + X10 + X11 + X12 + X13 + X14 + X15, data=dataset)
summary(fit)
parms

vif(fit)

# Then remove parameters with little influence
regFormula = paste('y ~', paste(names(dataset)[c(-1,-8,-9,-5,-16)], collapse=' + '))
fit = lm(regFormula, data=dataset)
```

```
# Same as
fit = lm(y ~ X1 + X2 + X3 + X5 + X6 + X9 +
X10 + X11 + X12 + X13 + X14,
data=dataset)
summary(fit)
parms

vif(fit)

# Now we take out the variables that are
contributing the least to the model

fit = lm(y ~ X1 + X2 + X3 + X5 + X6 + X9 +
X10 + X11 + X12, data=dataset)
summary(fit)
parms

vif(fit)
```

Regularized Manual Ridge

```
library(glmnet)
library(corrplot)
library(MASS)

# First we will do a little comparison of
# Maual regularization, the test dataset for glmnet
# is a nice small example to start with

load("QuickStartExample.RData")

# This dataset has an x-variable set with twenty columns
# and a y-variable set with one column

head(x)
head(y)

ds = as.data.frame(x)
dim(ds)
ds$y = y

# now reorder the columns
ds = ds[, c(21, 1:20)]
corrplot(cor(ds), method="ellipse")
head(ds)
```

```
# First let's let R do its job automatically

fitAuto = lm(y ~ ., data=ds[1:50, ])  # train on the first 50
summary(fitAuto)
plot(fitAuto)

xTrain = ds[1:50, 2:21]
yTrain = ds[1:50, 1]
yHat = predict(fitAuto)

# Note: degrees of freedom for residuals is 29 = 50 - $variables - 1
resid = yTrain - yHat
rse = sqrt(sum(resid^2) / 29)
rse  # Same as in summary!
```

cbind: Take a sequence of vector, matrix or data-frame arguments and combine by columns or rows, respectively.

ginv: returns an arbitrary generalized inverse of the matrix A, using gaussianElimination

```
# Now, lets to it manually
# Make a training set of the first and append the column of ones
xTrain = x[1:50, ]
xTrain = cbind(rep(1, 50), xTrain)
yTrain = y[1:50, ]

# Do the same thing for the test set
xTest = x[51:100, ]
xTest = cbind(rep(1, 50), xTest)
yTest = y[51:100, ]

# Try an initial fit
beta = ginv(t(xTrain) %*% xTrain) %*% t(xTrain) %*% yTrain
beta  # Same as fitAuto above

# Residual error
yHat = xTrain %*% beta
rse = sqrt(sum((yTrain - yHat)^2) / 29)
rse

# Now, get the prediction values for the test set
yPredict = xTest %*% beta
rse_predict = sqrt(sum((yTest - yPredict)^2) / 29)

# Notice how much bigger this is!
rse_predict
```

```
# Now, let's try a ridge regression, for example sake,
# try lambda = 1
lambda = 1
Ident = diag(ncol(xTrain))

betaRidge = ginv(t(xTrain) %*% xTrain + lambda * Ident) %*%
t(xTrain) %*% yTrain
betaRidge

# Residual error
yHat = xTrain %*% betaRidge
rseRidge = sqrt(sum((yTrain - yHat)^2) / 29)
rseRidge

# Now, get the prediction values for the test set
yPredict = xTest %*% betaRidge
rse_predict_ridge = sqrt(sum((yTest - yPredict)^2) / 29)

# Notice how the value is slightly better
rse_predict_ridge
rse_predict
```

```
# Just need to do a search for the best cross validated fit!
l = (1:100) / 10
b = array(0, c(21, 100))
rse = array(0, 100)
for (i in 1:length(l))
{
  b[,i] = ginv(t(xTrain) %*% xTrain + (l[i] * Ident)) %*% (t(xTrain)
%*% yTrain)
  yPredict = xTest %*% b[, i]
  rse[i] = sqrt(sum((yTest - yPredict)^2) / 29)
}

library(ggplot2)
qplot(l, rse)
rse

# Which lambda gives the minimum prediction rmse?
minI = which(rse == min(rse), arr.ind=T)
l[minI]
rse[minI]

# Best model
b[, minI]
```

glmnet package

```
#Load the package
library(glmnet)

#Import the data
data(QuickStartExample)

#Fit the model using the most basic call to glmbet
fit=glmnet(x,y)
print(fit)

#Plot the model
plot(fit)

#Obtain the actual coefficients at one or more lambda's
coef(fit, s=0.1)

#Use cv.glmnet to do cross validation
cvfit=cv.glmnet(x,y)

#Plot the object
plot(cvfit)

#View the selected lambda's and the corresponding coefficients
cvfit$lambda.min

#View the other selected lambda's
coef(cvfit, s="lambda.min")

#Make prediction
predict(cvfit, newx=x[1:5,], s="lambda.min")
```

Linear Regression

lambda.min: the value of lambda that gives minimum mean cross-validation error of the minimum

lambda.1se: gives the most regularized model such that error is within one standard error of the minimum

```
#Fit the model
fit=glmnet(x,y, alpha=0.2, weights=c(rep(1,50), rep(2,50)), nlambda=20)

#Print the glmnet object
print(fit)

#Plot fit
plot(fit, xvar="lambda", label=TRUE)

#Plot fit the other model
plot(fit, xvar="dev", label=TRUE)

#Extract the coefficients and make predictions at certain values of lambda
any(fit$lambda==0.5)
coef.exact=coef(fit, s=0.5, exact=TRUE)
coef.apprx=coef(fit, s=0.5, exact=FALSE)
cbind2(coef.exact, coef.apprx)

#Make predictions
predict(fit, newx=x[1:5,], type="response", 0.05)

#Use customize K-fold cross-validation
cvfit=cv.glmnet(x,y, type.measure = "mse", nfolds = 20)
```

```
#Parallel computing
library(iterators)
library(parallel)
library(doMC)

registerDoMC(cores=2)
X=matrix(rnorm(1e4*200), 1e4, 200)
y=rnorm(1e4)
system.time(cv.glmnet(X,Y))
system.time(cv.glmnet(X,Y, parallel=TRUE))

#lambda.min
cvfit$lambda.min
coef(cvfit, s="lambda.min")

#Make predictions
predict(cvfit, newx=x[1:5,], s="lambda.min")

#Users can control the folds used
foldid=sample(1:10, size=length(y), replace=TRUE)
cv1=cv.glmnet(x,y, foldid=foldid, alpha=1)
cv.5=cv.glmnet(x,y,foldid=foldid, alpha=.5)
cv0=cv.glmnet(x,y, foldid=foldid, alpha=0)

#Built-in plot functions
par(mfrow=c(2,2))
plot(cv1)
plot(cv.5)
plot(cv0)
plot(log(cv1$lambda), cv1$cvm, pch=19, col="red",xlab="Log(Lambda)",ulab=cv1$name)
points(log(cv.5$lambda), cv.5$cvm, pch=19, col="grey")
points(log(cv0$lambda), cv0$cvm, pch=19, col="blue")
legend("topleft", legend=c("alpha=1", "alpha=.5", "alpha 0"), pch=19,
col=c("red","grey","blue"))

#Coefficient upper and lower bounds
tfit=glmnet(x,y,lower=-.7, upper=.5)
plot(tfit)
```

Penalty Factors

```
p.fac=rep(1,20)
p.fac[c(5,10,15)]=0
pfit=glmnet(x,y,
penalty.factor=p.fac)
plot(pfit, label=TRUE)
```

Customizing Plot

```
set.seed(101)
x=matrix(rnorm(1000), 100, 10)
y=rnorm(100)
vn=paste("var", 1:10)
fit=glmnet(x,y)
plot(fit)
par(mar=c(4.5,4.5,1.4))
plot(fit)
vnat=coef(fit)
#Remove the intercept and get the coefficients at the end of the path
vnat=vnat[-1,ncol(vnat)]
axis(4, at=vnat, line=-.5, label=vn, las=1, tick=FALSE, cex.axis=0.5)
```

Multi-response Gaussian Family

```
#Load the dataset
data("MultiGaussianExample.RData")

#fit the data
mfit=glmnet(x,y, family="mgaussian")

#Plot the coefficients
plot(mfit, xvar="lambda", label=TRUE, type.coef="2norm")

#Make predictons
predict(mfit, newx=x[1:5,], s=c(0., 0.01))

#K-fold cross-validation
cvmfit=cv.glmnet(x,y, family="magaussian")

#Plot the resulting
plot(cvmfit)

#Show explicitly the selected optiaml values of lambda
cvmfit$lambda.min
cvmfit$lambda.1se
```

Logistic Regression

```
#Load the data
data(BonomialExample)

#Fit the data
fit=glmnet(x,y, family="binomial")

#Plot the result
plot(fit, xvar="dev", label=TRUE)

#Make predictions
predict(fit, newx=x[1:5,], type="class", s=c(0.05, 0.01))

#Cross vaidation
cvfit=cv.glmnet(x,y, family="binomial", type.measure="class")

#plot the object
plot(cvfit)

cvfit$lambda.min
cvfit$lambda.1se

#Review by some example
coef(cvfit, s="lambda.min")
predict(cvfit, newx=x[1:10,], s="lambda.min", type="class")
```

Multinomial Models

```
#Load the data
data(MultiNomialExample)

#Fit the model
fit=glmnet(x,y, family="multinomial",
type.multinomial="grouped")

#Plot the fit
plot(fit, xvar="lambda", label=TRUE,
type.coef="2norm")

#Cross-validation and plot the returned object
cvfit=cv.glmnet(x, y, family="multinomial",
type.multinomial="grouped", parallel=TRUE)
plot(cvfit)

#Predict at the optimally selected lambda
predict(cvfit, newx=x[1:10,], s="lambda.min",
type="class")
```

Poisson Models

```
#Load the data
data(poissonExample)

#Apply glmnet
fit=glmnet(x,y, family="poisson")

#Plot the ift
plot(fit)

#Extrat the coefficients
coef(fit, s=1)

#Make predictions
predict(fit, newx=x=[1:5,], type="response", s=c(0.1,1))

#Use cross-validation
cvfit=cv.glmnet(x,y,family="poisson")

#Plot the object
plot(cvfit)

#Show the optimal lambda's and the corresponding coefficients
opt.lam=c(cvfit$lambda.min, cvfit$lambda.1se)
coef(cvfit, s=opt.lam)
```

Cox Models

```
#Load the data
data(CoxExample)
y[1:5,]

#Apply the glmnet function
fit=glmnet(x,y,family="cox")

#Plot the coefficients
plot(fit)

#Extract the coefficients at certain values of lamba
coef(fit, s=0.05)

#Apply cross validation
cvfit=cv.glmnet(x,y,family="cox")

#Plot the object
plot(cvfit)

#Extract certain lmanda
cvfit$lambda.min
cvfit$lambda.1se

#Check the active covariates in our model and see their coefficients
coef.min=coef(cvfit, s="lambda.min")
active.min=which(coef.min != 0)
index.min=coef.min[active.min]
index.min
coef.min
```

Sparse Matrices

```
#Load the data
data(SparseExample)

#Response vector
class(x)

#Fit the model the same way
fit=glmnet(x,y)

#Do the cross-validation and plot the resulting object
cvfit=cv.glmnet(x,y)
plot(cvfit)

#Create predict function
i=sample(1:5, size=25, replace=TRUE)
j=sample(1:20, size=25, replace=TRUE)
x=rnorm(25)
nx=sparseMatrix(i=i, j=j, x=x, dims=c(5,20))
predict(cvfit, newx=nx, s="lambda.min")
```

Analysis of Variance

ANOVA()

anova()

Checking the Assumptions of Linear Regression

Checking the Assumption of Linearity

The relationship between all X's and Y is linear

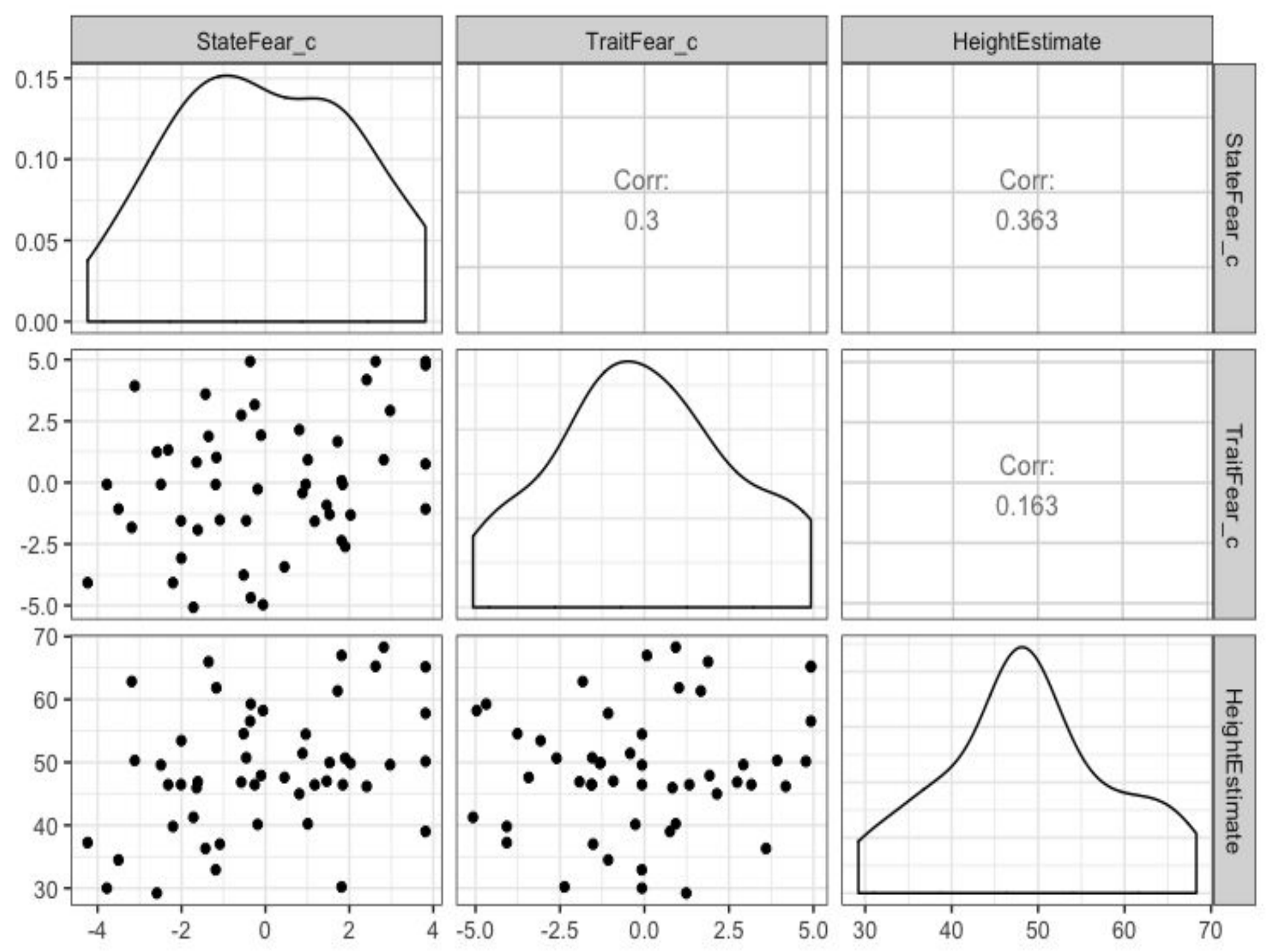
Violation of this assumption leads to changes in. regression coefficient

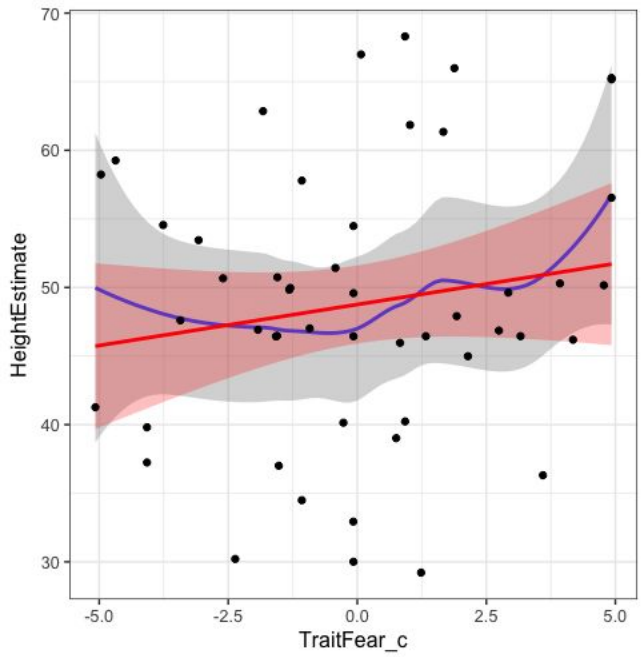
Scatterplot

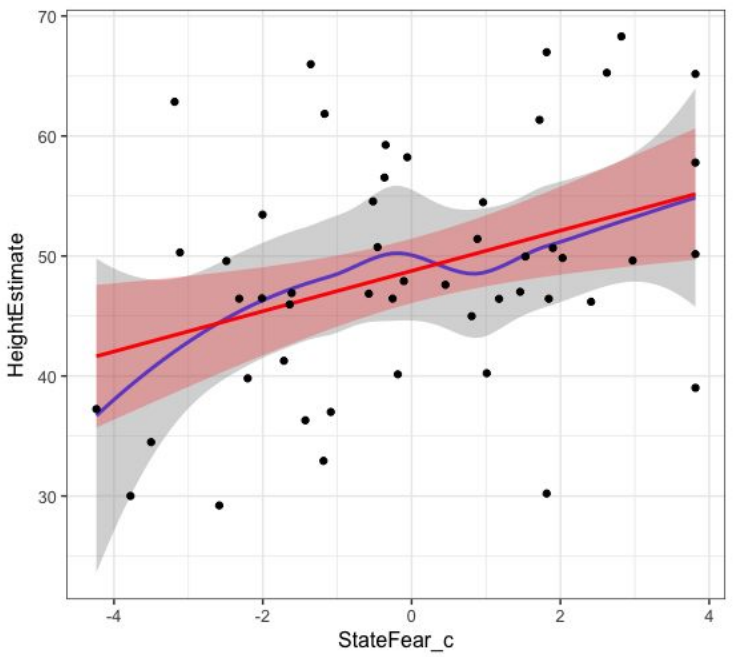
Loess smoother:

The loess smoother roughly approximates the linear line, the assumption of linearity has been met

```
1  library(ggplot2)
2  library(GGally)
3  library(car)
4  library(MASS)
5
6  feardata=read.csv(url('http://www.ianruginski.com/files/feardata.csv'))
7  fit=lm(HeightEstimate~StateFear_c+TraitFear_c, feardata)
8
9  #Checking the assumption of linearity
10 theme_set(theme_bw(base_size = 12))
11 ggpairs(feardata, columns=7:5)
12
```


StateFear_c
TraitFear_c
HeightEstimate
StateFear_c
TraitFear_c
HeightEstimate
Corr:
0.3
Corr:
0.363
Corr:
0.163
0.15
0.10
0.05
0.00
5.0
2.5
0.0
-2.5
-5.0
70
60
50
40
30
-4
-2
0
2
4
-5.0
-2.5
0.0
2.5
5.0
30
40
50
60
70






```
ggplot(feardata, aes(TraitFear_c, HeightEstimate))+stat_smooth(method="loess")+stat_smooth(method="lm", color="red", fill="red", alpha=.25)+geom_point()
```

```
ggplot(feardata, aes(StateFear_c, HeightEstimate))+stat_smooth(method="loess")+stat_smooth(method="lm", color="red", fill="red",alpha=.25)+geom_point()
```

In these plots, the grey shading corresponds to the standard errors for the loess smoother fit, while the red shading corresponds to the standard errors for the linear fit. The blue line is a loess smoothed line and the red line is a linear regression line. As long as the loess smoother roughly approximates the linear line, the assumption of linearity has been met. It appears that the relationships between X's and Y's are roughly linear.

Constant Variance of Residuals

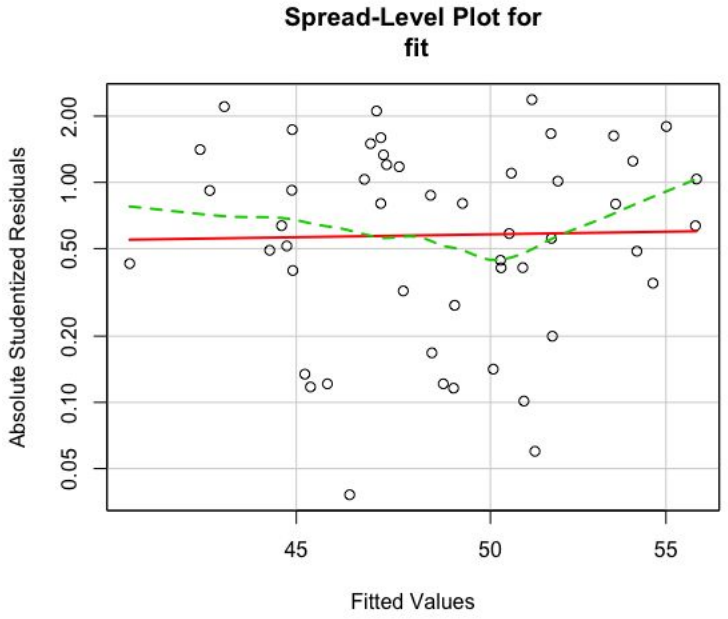
spreadLevelPlot(fit)

A red line (linear fit) and a dashed green line (loess smoothed fit)

We are looking for any lawful curves or skewness in the data that may suggest that our regression model is better or worse at predicting for specific levels of our predictors

Absolute studentized residuals refers to the absolute values (ignoring over or underfitting) of the quotient resulting from the division of a residual by an estimate of its standard deviation

These should be roughly equally distributed across the range of the fitted y values



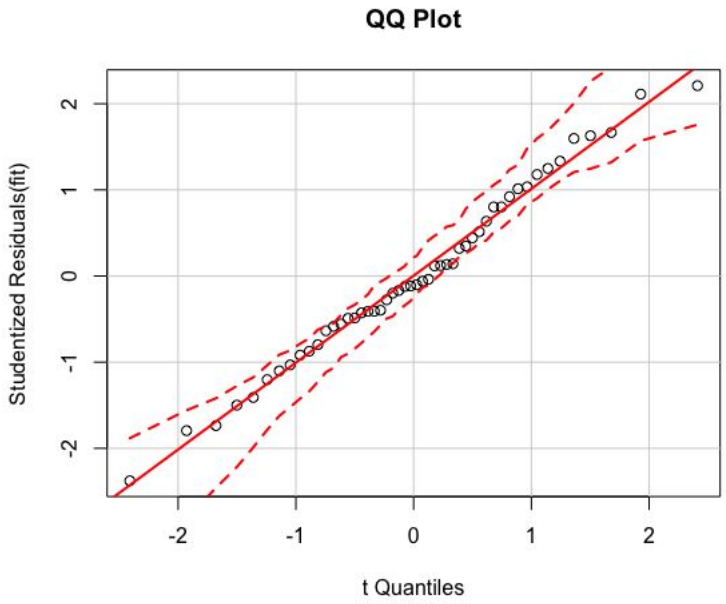
Normality of Residuals

qqPlot(fit, main="QQ plot")

Q-Q plots are used to assess whether your distribution of residuals (represented on the Y axis) roughly approximate a normal distribution of residuals (represented on the X axis)

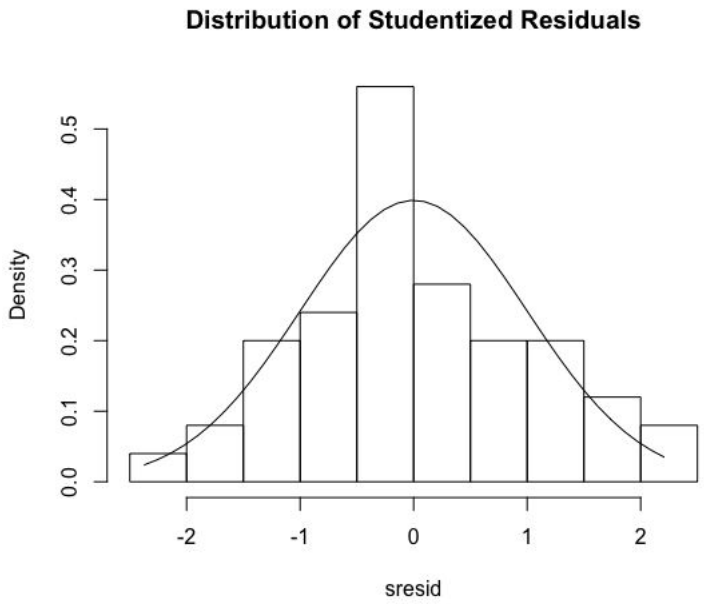
The points should mostly fall on the diagonal line in the middle of the plot

If the assumption is violated, the points will fall in some sort of curve shape, or will form two separate variable lines



Normality of Residuals

we overlay a normal distribution curve on top of the histogram, and observe that the histogram roughly approximates a normal distribution



Check for Multicollinearity

```
> vif(fit)
StateFear_c TraitFear_c
   1.098924    1.098924
> sqrt(vif(fit))
StateFear_c TraitFear_c
   1.048296    1.048296
```

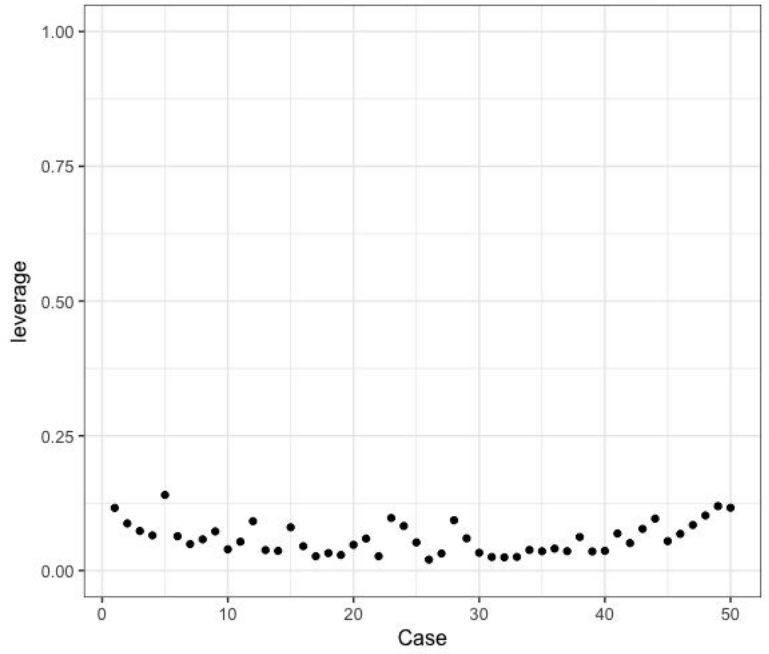
The lower the VIF, the better. If the VIF is less than 2, we meet the assumption of no multicollinearity.

Check for outliers

Leverage

we are looking for any point that visually sticks out. In this case, it doesn't appear that any points exhibit abnormally high leverage.

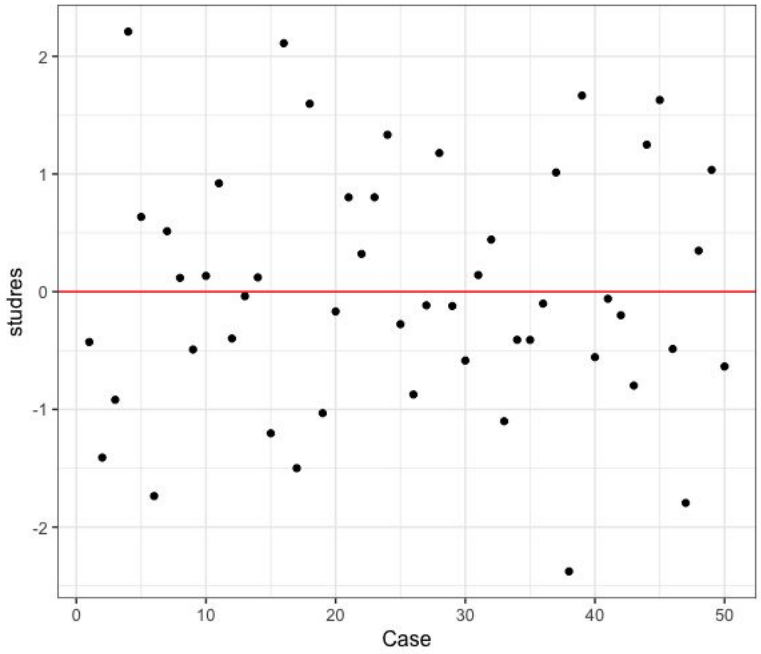
```
> #Check for outliers--Leverage
> feardata$leverage=hatvalues(fit)
> ggplot(feardata, aes(X, leverage))+geom_point()+ylim(0,1
)+xlab('Case')
```



Check for Outliers

Discrepancy

Amount of difference between observed and predicted values. Main indicator is called externally studentized residual. These are residuals calculated after removing each case&rerunning the regression.

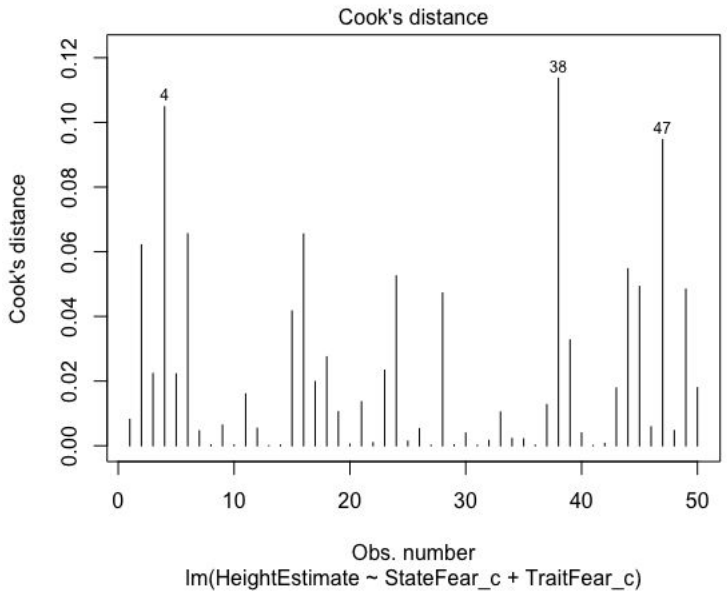



```
> #Check for outliers--Discrepancy
> feardata$studres=studres(fit)
> ggplot(feardata, aes(X, studres))+geom_point()+geom_hlin
e(color="red", yintercept = 0)+xlab('Case')
>
```

Check for Outliers

Influence

A combination of leverage and discrepancy. Global indicators of influence include Difference in fits, standardized (DFFITS)&Cook's distance, and indicate how much Yhat (predicted Y) changes based on removal of a case. Specific indicators of influence (Difference in fits of betasindicate how much specific regression coefficients in your model would change

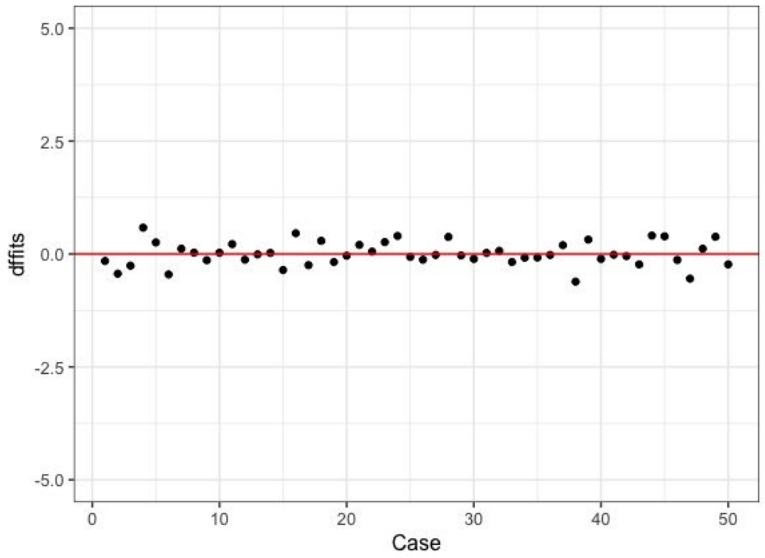



```
> #Identify D values > 4/(n-k-1)
> cutoff=4/((nrow(feardata)-length(fit$coefficients)-2))
> plot(fit, which=4, cook.levels = cutoff)
```

Check for Outliers

DFFITS

Though these should be very similar to Cook's distance, its good to be thorough and check multiple indicators of global influence.

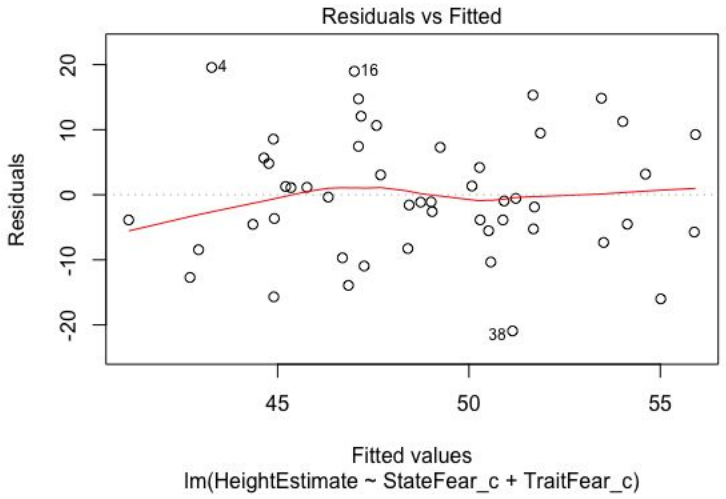



```
> feardata$dffits=dffits(fit)
> ggplot(feardata, aes(X, dffits))+geom_point()+geom_hline(color="red", yintercept = 0)+xlab('Case')+ylim(-5,5)
```

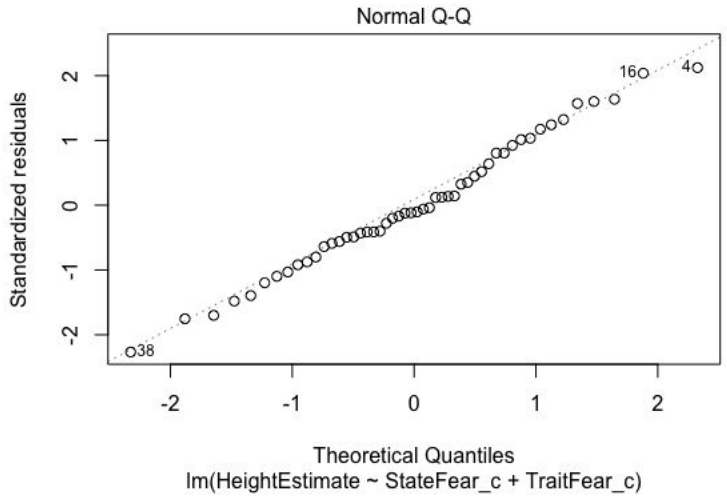
Checking many assumptions at once by plotting

plot(fit)

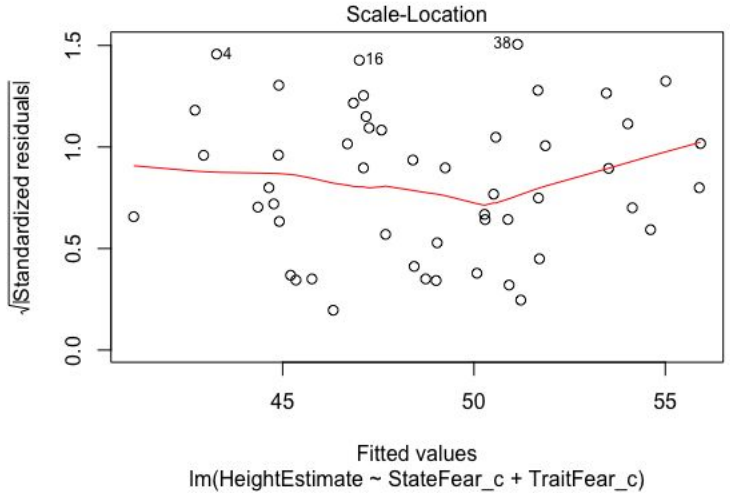
Residuals vs Fitted Values, to check constant variance in residuals and linearity of association between predictors and outcome (look for a relatively straight line and random-looking scatterplot).



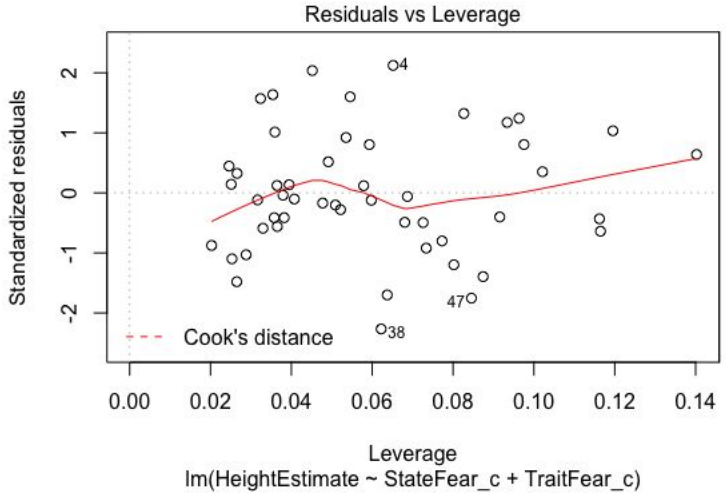
Normal Q-Q Plot, to check the assumption of normally distributed residuals.



Root of Standardized residuals vs Fitted values, this is very similar to number 1, where the Y axis of residuals is in a different metric



Residuals vs Leverage, to check if the leverage of certain observations are driving abnormal residual distributions, thus violating assumptions and biasing statistical tests. Potentially problematic points will be labelled in the plot.



Checking many assumptions at once by function

library(gvlma)
gvlma(fit)

Are the relationships between your X predictors and Y roughly linear? Rejection of the null (p<.05) indicates a non-linear relationship between one or more of your X's and Y. This means that you should likely use an alternative modeling technique or add an additional transformed X term to your model to agree with the data structure (e.g. add a quadratic term, X-squared, to the model if the relationship seems curvilinear from further scatterplot inspection).

Is your distribution skewed positively or negatively, necessitating a transformation to meet the assumption of normality? Rejection of the null (p<.05) indicates that you should likely transform your data.

Is your distribution kurtotic (highly peaked or very shallowly peaked), necessitating a transformation to meet the assumption of normality? Rejection of the null (p<.05) indicates that you should likely transform your data.

Is your dependent variable truly continuous, or categorical? Rejection of the null (p<.05) indicates that you should use an alternative form of the generalized linear model (e.g. logistic or binomial regression).

Is the variance of your model residuals constant across the range of X (assumption of homoscedastiity)? Rejection of the null (p<.05) indicates that your residuals are heteroscedastic, and thus non-constant across the range of X. Your model is better/worse at predicting for certain ranges of your X scales.

```
> library(gvlma)
> gvlma(fit)

Call:
lm(formula = HeightEstimate ~ StateFear_c + TraitFear_c, data = feardata)

Coefficients:
(Intercept)  StateFear_c  TraitFear_c
    48.7580       1.5965       0.2177


ASSESSMENT OF THE LINEAR MODEL ASSUMPTIONS
USING THE GLOBAL TEST ON 4 DEGREES-OF-FREEDOM:
Level of Significance =  0.05

Call:
 gvlma(x = fit)

                     Value p-value
Global Stat        0.70351  0.9509
Skewness           0.11302  0.7367
Kurtosis           0.37881  0.5382
Link Function      0.17365  0.6769
Heteroscedasticity 0.03803  0.8454
                                 Decision
Global Stat        Assumptions acceptable.
Skewness           Assumptions acceptable.
Kurtosis           Assumptions acceptable.
Link Function      Assumptions acceptable.
Heteroscedasticity Assumptions acceptable.
```

Data Analysis Case

Real Estate Data Analysis

```
# Load a library with some extra linear algebra functions
including a matrix inverse
library("MASS")

data =
read.table("C:/Users/sbesser/Downloads/real-estate.csv
", sep=",", header=T)

lot_size = data$Lot
house_sqft = data$Sqft
sold = data$Price
```

```
#################################################
#####
# First run a fit based with the build-in linear
# model function
#################################################
#####
fit = lm(sold ~ lot_size + house_sqft)

summary(fit)

newdata = data.frame(lot_size=c(4), house_sqft=c(3))
predict.lm(fit, newdata, interval="confidence", level=.99)
```

```
# We pull out the dependent and independent variables
and convert them to matrices
X = data.matrix(data[, 3:4])            # pull out the
independent variables (columns 3 and 4)
Y = data.matrix(data[, 2])              # pull out the
depedent variable (column 2)

# Get the number of samples and the number of
independent variables for
# computing the degrees of freedom later
nSamples = nrow(X)
nInd = ncol(X)

# Now add a column of ones to the X's to create the
matrix for regression
Z = cbind(rep(1, nSamples), X)



# Here, we manually calculate some of the regression
# parameters to exercise R's matrix functionality
```

```
# We pull out the dependent and independent variables
and convert them to matrices
X = data.matrix(data[, 3:4])              # pull out the
independent variables (columns 3 and 4)
Y = data.matrix(data[, 2])               # pull out the
depedent variable (column 2)

# Get the number of samples and the number of
independent variables for
# computing the degrees of freedom later
nSamples = nrow(X)
nInd = ncol(X)
```

```
# Now add a column of ones to the X's to create the
matrix for regression
Z = cbind(rep(1, nSamples), X)

# Comput (Z^T * Z)^-1 * Z^T * Y
beta = ginv(t(Z) %*% Z) %*% t(Z) %*% Y
print("Coefficients:")
print(beta[,1], digits=1)

# Compute the model's estimates for Y
Y_Hat = Z %*% beta

# Now compute the residuals
e = Y - Y_Hat
```

```
# Now compute the error variance, note that there are
"nInd + 1" parameters
# and so "nSamples - (nInd + 1)" degrees of freedom
s = sqrt( (t(e) %*% e) / (nSamples - (nInd + 1)) )
s = s[1,1]
print("Error Std Deviation")
print(s)
# Now compute the R^2.  Remember, this is the fitted
variance sum of squares
# divided by the original variance sum of squares.  It
gives the percentage
# of the Y's variance that is explained by the fit
yMean = mean(Y)

R2 = sum((Y_Hat - yMean)^2) / sum((Y - yMean)^2)
print("Coefficient of Determination (R^2)")
print(R2, digits=2)
```

```
# Now, find the F-score of the beta's
explainedVariance = sum((Y_Hat - yMean)^2) / nInd  #
Degrees of freedom = #beta - 1
unexplainedVariance = sum((Y - Y_Hat)^2) / (nSamples -
nInd - 1)
f = explainedVariance / unexplainedVariance

CovMatrix = s^2 * ginv(t(Z) %*% Z)
print(CovMatrix)

betaVar = c(sqrt(CovMatrix[1,1]), sqrt(CovMatrix[2,2]),
sqrt(CovMatrix[3,3]))
print(betaVar)
```

PCA & PCA Plot

```
library(foreign)  # Allows us to read spss files!
library(corrplot)
library(car)
library(QuantPsyc)
library(leaps)
```

PCA Plot:

```
# PCA_Plot functions
PCA_Plot = function(pcaData)
{
  library(ggplot2)

  theta = seq(0,2*pi,length.out = 100)
  circle = data.frame(x = cos(theta), y = sin(theta))
  p = ggplot(circle,aes(x,y)) + geom_path()

  loadings = data.frame(pcaData$rotation, .names =
row.names(pcaData$rotation))
  p + geom_text(data=loadings, mapping=aes(x = PC1, y = PC2,
label = .names, colour = .names, fontface="bold")) +
    coord_fixed(ratio=1) + labs(x = "PC1", y = "PC2")
}
```

```
PCA_Plot_Secondary = function(pcaData)
{
  library(ggplot2)

  theta = seq(0,2*pi,length.out = 100)
  circle = data.frame(x = cos(theta), y =
sin(theta))
  p = ggplot(circle,aes(x,y)) + geom_path()

  loadings = data.frame(pcaData$rotation,
.names = row.names(pcaData$rotation))
  p + geom_text(data=loadings,
mapping=aes(x = PC3, y = PC4, label =
.names, colour = .names,
fontface="bold")) +
    coord_fixed(ratio=1) + labs(x = "PC3", y
= "PC4")
}
```

```
PCA_Plot_Psyc = function(pcaData)
{
 library(ggplot2)

 theta = seq(0,2*pi,length.out = 100)
 circle = data.frame(x = cos(theta), y = sin(theta))
 p = ggplot(circle,aes(x,y)) + geom_path()

 loadings = as.data.frame(unclass(pcaData$loadings))
 s = rep(0, ncol(loadings))
 for (i in 1:ncol(loadings))
 {
  s[i] = 0
  for (j in 1:nrow(loadings))
    s[i] = s[i] + loadings[j, i]^2
  s[i] = sqrt(s[i])
 }

 for (i in 1:ncol(loadings))
  loadings[, i] = loadings[, i] / s[i]
```

```
loadings$.names = row.names(loadings)

  p + geom_text(data=loadings, mapping=aes(x = PC1, y = PC2, label = .names, colour =
.names, fontface="bold")) +
    coord_fixed(ratio=1) + labs(x = "PC1", y = "PC2")
}

PCA_Plot_Psyc_Secondary = function(pcaData)
{
  library(ggplot2)

  theta = seq(0,2*pi,length.out = 100)
  circle = data.frame(x = cos(theta), y = sin(theta))
  p = ggplot(circle,aes(x,y)) + geom_path()

  loadings = as.data.frame(unclass(pcaData$loadings))
  s = rep(0, ncol(loadings))
  for (i in 1:ncol(loadings))
  {
    s[i] = 0
    for (j in 1:nrow(loadings))
      s[i] = s[i] + loadings[j, i]^2
    s[i] = sqrt(s[i])
  }

  for (i in 1:ncol(loadings))
    loadings[, i] = loadings[, i] / s[i]

  loadings$.names = row.names(loadings)

  print(loadings)
  p + geom_text(data=loadings, mapping=aes(x = PC3, y = PC4, label = .names, colour =
.names, fontface="bold")) +
    coord_fixed(ratio=1) + labs(x = "PC3", y = "PC4")
}
```

```
# Read in the hbat spss "hbat" dataset from the book by Hair, et. al.
hbat = read.spss("HBAT.sav", to.data.frame=T)
head(hbat)

# Pull out just the numeric fields and place customer satisfaction
# at the front because it will be our "Y".  It makes it easier to
# interpret correlation matrices!
hbatNumeric = hbat[, c(20, 7:19)]
head(hbatNumeric)
plot(hbatNumeric)

# Compute the correlation matrix and visualize it
cor.hbat = cor(hbatNumeric)
cor.hbat
corrplot(cor.hbat, method="ellipse")

# Look at the size of the numeric data
dim(hbatNumeric)
```

```
# Run a correlation test to see how correlated the variables are.  Which correlations are significant
library(psych)
options("scipen"=100, "digits"=5)
round(cor(hbatNumeric[, 2:14]), 2)
MCorrTest = corr.test(hbatNumeric[, 2:14], adjust="none")
MCorrTest

M = MCorrTest$p
M

# Now, for each element, see if it is < .01 (or whatever significance) and set the entry to
# true = significant or false
MTest = ifelse(M < .01, T, F)
MTest

# Now lets see how many significant correlations there are for each variable.  We can do
# this by summing the columns of the matrix
colSums(MTest) - 1  # We have to subtract 1 for the diagonal elements (self-correlation)
```

```
# We remove two columns, the first (x15 = column 11) is not correlated with anything else
# The second (x17 = column 13) is correlated with too many other variables
hbatReduced = hbatNumeric[, -c(1, 11, 13)]
head(hbatReduced)
cor(hbatReduced)

p = prcomp(hbatReduced, center=T, scale=T)
plot(p)
abline(1, 0)
summary(p)
print(p)
par(mar=c(2, 2, 2, 2))
plot(p)
PCA_Plot(p)
PCA_Plot_Secondary(p)
biplot(p)

rawLoadings = p$rotation %*% diag(p$sdev, nrow(p$rotation), nrow(p$rotation))
print(rawLoadings)
v = varimax(rawLoadings)
ls(v)
v
```

```
# The Psych package has a wonderful PCA function that allows many more options
# including build-in factor rotation, specifiying a number of factors to include
# and automatic "score" generation
p2 = psych::principal(hbatReduced, rotate="varimax", nfactors=4, scores=TRUE)
print(p2$loadings, cutoff=.4, sort=T)
p2$loadings
p2$values
p2$communality
p2$rot.mat

v$loadings

PCA_Plot_Psyc(p2)
PCA_Plot_Psyc_Secondary(p2)

fit = factanal(hbatReduced, 4)
print(fit$loadings, cutoff=.4, sort=T)
summary(fit)

par(mar=c(4, 4, 4, 4))
plot(p)
abline(1, 0)

# Finally, lets plot the principal components here
PCA_Plot(p)
```

PCA Plot & PCA

Iris Dataset

```
PCA_Plot = function(pcaData)
{
  library(ggplot2)

  theta = seq(0,2*pi,length.out = 100)
  circle = data.frame(x = cos(theta), y =
sin(theta))
  p = ggplot(circle,aes(x,y)) + geom_path()

  loadings = data.frame(pcaData$rotation,
.names = row.names(pcaData$rotation))
  p + geom_text(data=loadings,
mapping=aes(x = PC1, y = PC2, label =
.names, colour = .names,
fontface="bold")) +
    coord_fixed(ratio=1) + labs(x = "PC1", y
= "PC2")
}
```

```
PCA_Plot_Varimax = function(v)
{
 library(ggplot2)

 theta = seq(0,2*pi,length.out = 100)
 circle = data.frame(x = cos(theta), y = sin(theta))
 p = ggplot(circle,aes(x,y)) + geom_path()

 loadings = as.data.frame(unclass(v$loadings))
 s = rep(0, ncol(loadings))
 for (i in 1:ncol(loadings))
 {
  s[i] = 0
  for (j in 1:nrow(loadings))
   s[i] = s[i] + loadings[j, i]^2
  s[i] = sqrt(s[i])
 }
for (i in 1:ncol(loadings))
  loadings[, i] = loadings[, i] / s[i]

 loadings$.names = row.names(loadings)

 p + geom_text(data=loadings, mapping=aes(x = V1, y = V2, label =
.names, colour = .names, fontface="bold")) +
  coord_fixed(ratio=1) + labs(x = "PC1", y = "PC2")
}
```

```
PCA_Plot_Psyc = function(pcaData)
{
  library(ggplot2)

  theta = seq(0,2*pi,length.out = 100)
  circle = data.frame(x = cos(theta), y =
sin(theta))
  p = ggplot(circle,aes(x,y)) + geom_path()

  loadings = data.frame(pcaData$rot.mat,
.names = row.names(pcaData$loadings))
  print(loadings)
  p + geom_text(data=loadings,
mapping=aes(x = X1, y = X1, label =
.names, colour = .names,
fontface="bold")) +
    coord_fixed(ratio=1) + labs(x = "PC1", y
= "P2")
}
```

PCA on three components of the iris data

```
library(ggplot2)
library(GGally)

data = iris[, c(1, 3, 4)]
ggpairs(data)
p = prcomp(data)

p
summary(p)

options(scipen=100, digits=2)
R = p$rotation
rotatedData =
as.data.frame(as.matrix(data) %*% R)
round(cov(rotatedData), digits=2)
ggpairs(rotatedData)
```

Experimenting with units

```
# Make a copy of the dataset and change the units to mm instead of cm (x10)

irisOrig = iris[, 1:4]    # just the numeric stuff
irisMod = irisOrig
irisMod$Petal.Length = irisMod$Petal.Length * 10

ggpairs(irisMod)

pOrig = prcomp(irisOrig)
pMod = prcomp(irisMod)

print(pMod)    # The first component is now entirely "Petal.Length"
print(pOrig)

summary(pMod)

pModScale = prcomp(irisMod, scale=T)
summary(pModScale)
print(pModScale)

PCA_Plot(pModScale)
plot(pModScale)
```

Data Analysis Case

HBAT Case

```
library(foreign)  # Allows us to read spss files!
library(corrplot)
library(car)
library(QuantPsyc)
library(leaps)

# Read in the hbat spss "hbat" dataset from the book by Hair, et. al.
hbat = read.spss("HBAT.sav", to.data.frame=T)
head(hbat)

# Pull out just the numeric fields and place customer satisfaction
# at the front because it will be our "Y".  It makes it easier to
# interpret correlation matrices!
hbatNumeric = hbat[, c(20, 7:19)]
head(hbatNumeric)
plot(hbatNumeric)
```

```
# Compute the correlation matrix and visualize it
cor.hbat = cor(hbatNumeric)
cor.hbat
corrplot(cor.hbat, method="ellipse")

# Try a fit of the full set of parameters.  Note that the . in
# the regression formula gives all the rest of the parameters.
fullFit = lm(x19 ~ . - x14 - x18, data=hbatNumeric)
summary(fullFit)

# And compute the vif scores to get an idea of the multicolinearities here!
vif(fullFit)

# Now, fit with a single parameter (the one with the highest correlation)
fit1 = lm(x19 ~ x9, data=hbatNumeric)
summary(fit1)

# Get the standardized coefficients for x9
lm.beta(fit1)

# Try adding another parameter (this one has the most correlation with
the residuals of the last fit)
fit2 = lm(x19 ~ x9 + x6, data=hbatNumeric)
summary(fit2)
vif(fit2)
```

Automated fitting

leaps() performs an exhaustive search for the best subsets of the variables in x for predicting y in linear regression

scale: which summary statistic to use for ordering plots

```
# The leaps package has a beautiful subset search routine that also provides a viaualization
# of its results
hbatSubsets = regsubsets(x19 ~ x6 + x7 + x8 + x9 + x10 + x11 + x12 + x13 + x14 + x15 + x16 + x17 + x18, data=hbatNumeric, nbest=10)


plot(hbatSubsets, scale="adjr2")


bestR2Fit = lm(x19 ~ x6 + x7 + x9 + x11 + x12 + x16, data=hbatNumeric)
summary(bestR2Fit)


# The R function "step" can perform stepwise regression, but to get going, we need to feed
# it the two "bounding" models so that it knows what to work with
null = lm(x19 ~ 1, data=hbatNumeric)
null

full = lm(x19 ~ ., data=hbatNumeric)
full
```

```
# First we do a forward search
hbatForward = step(null, scope = list(lower=null,
upper=full), direction="forward")
summary(hbatForward)
# Compare the results to the full search above

# Next do a backward search
hbatBackward = step(full, direction="backward")
summary(hbatBackward)

# Finally we do a "stepwise" search combining the
two
hbatStep = step(null, scope = list(upper=full),
direction="both")
summary(hbatStep)
summary(hbatForward)
summary(hbatBackward)

# Things are really nice if they all agree!

dim(hbatNumeric)
```

Option: Allow the user to set and examine a variety of global options which affect the way in which R computes and displays its results

scipen: integer. A penalty to be applied when deciding to print numeric values in fixed or exponential notation. Positive values bias towards fixed and negative towards scientific notation: fixed notation will be preferred unless it is more than scipen digits wider

```
library(psych)
options("scipen"=100, "digits"=2)
cor(hbatNumeric[, 2:14])
MCorrTest = corr.test(hbatNumeric[, 2:14], adjust="none")
MCorrTest

M = MCorrTest$p
M

# The probability matrix uses a different test above the diagonal, we are just interested in
# the entries below the diagonal, so we make it symmetric
for (i in 2:nrow(M))
  for (j in 1:(i-1))  # Only grab elements below the diagonal
    M[j, i] = M[i,j]  # Copy into the corresponding element above the diagonal
M

# Now, for each element, see if it is < .01 (or whatever significance) and set the entry to
# true = significant or false
MTest = ifelse(M < .01, T, F)
MTest

# Now lets see how many significant correlations there are for each variable.  We can do
# this by summing the columns of the matrix
colSums(MTest) - 1  # We have to subtract 1 for the diagonal elements (self-correlation)
```

PCA Case

Customer Survey

```
library(psych)

cust = read.csv("CustomerSurvey.csv", header=T)
head(cust)

# Grab just the customer questions.  The 47th is the overall rating of the airport as a whole
custQuest = cust[, c(47 ,34:46)]
head(custQuest)

# The plot shows some groups of correlations
library(corrplot)
corrplot(cor(custQuest))

round(cor(custQuest), 2)
test = corr.test(cor(custQuest), adjust="none")
M = test$p
round(M, 2)
MTest = ifelse(M < .01, T, F)
colSums(MTest) - 1  # We have to subtract 1 for the diagonal elements (self-correlation)
```

```
full = lm(Q7ALL ~ ., data=custQuest)
summary(full)

# Residuals show fairly typical results for survey data.  The "diagonal lines" in the
# residual plot are from the quantization of the answers ... might want to try excluding
# the zeros here because that is the ones that left this queston blank
plot(full)

null = lm(Q7ALL ~ 1, data=custQuest)

# No severe vif scores
library(car)
vif(fit)
```

```
# Let's try a little model building, this gives 9 parameters including restrooms,
# boarding, food, walkways, rentals and signage, etc.  Some parameters of marginal significance
library(leaps)
forward = step(null, scope = list(lower=null, upper=full), direction="forward")
summary(forward)

# The reverse model is the same, so we don't seem to be missing contributions here.
backward = step(full, direction="backward")
summary(backward)

par(mar=c(3, 3, 3, 3))
# Now, let's see what we get with a simple PCA
p = princomp(custQuest[, -1])
plot(p)
abline(1, 0)

# We'll try looking at the first 5, which capture 80 of the variance of the dataset
p
summary(p)

recast = as.data.frame(p$scores[, 1:5])
head(recast)
plot(recast)
```

```
# We don't quite get the same level of residual error (~2% more),
# but the model is quite a bit sparser
names(recast) = c("PC1", "PC2", "PC3", "PC4", "PC5")
recast$Q7ALL = custQuest$Q7ALL
fitPCA = lm(Q7ALL ~ ., data=recast)
summary(fitPCA)
plot(fitPCA)

# The question is, how can we interpret these PC's.  The "loadings"
# here show the makeup of each PC
p$loadings

# Now let's try rotating the components (factors)
library(psych)
p2 = principal(custQuest[, -1], nfactors=5)
print(p2$loadings, cutoff=.4)

# Notice that we do a bit better here than with the bare PCA
p2
recast = as.data.frame(p2$scores)
recast$Q7ALL = cust$Q7ALL
fitRot = lm(Q7ALL ~ ., data=recast)
summary(fitRot)
```

PCA Case

Mart

```
library(Amelia)    # Has one "missmap" function for finding missing values
library(ggplot2)
library(GGally)    # For the ggpairs function
library(corrplot)

# Load the dataset
mart = read.csv("Mart_Train.csv", header=T)
head(mart)

# Rename the columns for better viewing
names(mart) = c("ID", "Weight", "Fat", "Visibility", "Type", "MSRP", "Outlet", "O_Estab", "O_Size", "O_Loc", "O_Type", "Sales")
head(mart)
```

```
# Clean the dataset
# The data contains many missing values, let's investigate.
# There is one function for visualizing missing values ... all the missing values
# are in weight ... that helps
missmap(mart)

# Now, let's see if there is any dependence of these missing values on any
# of the other variables.  To do this we will make a temporary copy of the
# dataset and put a value in for the weight outside of the range
martTmp = mart
range(martTmp$Weight, na.rm=T)

# This fills in those missing values.  Note that the valid range is 4.5 ... 21.4
# So, replace these values with 50.
martTmp$Weight = replace(mart$Weight, is.na(mart$Weight) == T, 50)
missmap(martTmp)

# Check to see that there are no more missing values
missmap(martTmp)
```

```
# Now, let's graph the values ... for the Type, they are spread over the
# entire set of product types
ggplot(martTmp, aes(x=Type, y=Weight)) + geom_jitter() + coord_flip()

# But we get a different story when look at the Outlet. It looks like
# only two outlets are systematically not reporting weights
ggplot(martTmp, aes(x=Outlet, y=Weight)) + geom_jitter()

# Let's look at the ID #'s assuming that these identify a unique product
# We cast ID to a number so that the x-axis will be sensible (it won't
# change the presentation of the graph because the ordering will be the
# same)
ggplot(martTmp, aes(x=as.numeric(ID), y=Weight)) + geom_point()
```

```
# So, let's go into the original dataset and for each missing value, look
# for other stores that have that product ID.  Then we'll set this product
# ID to the average of those (We would actually expect that they are all the
# same)
newMart = mart
for (i in 1:nrow(newMart))
{
  if (is.na(newMart$Weight[i]))
  {
    prodID = newMart$ID[i]
    matching = newMart[newMart$ID == prodID, ]
    newMart$Weight[i] = mean(matching$Weight, na.rm=T)
  }
}

missmap(newMart)  # Looks like there is only one missing product now ... a vast improvement
```

```
# Removing incomplete cases
# Since there are only two products with missing weights now, we discard them
# The "complete.cases" function is designed for this
newMart = newMart[complete.cases(newMart), ]


# Replace the dataset with the cleaned one
mart = newMart
```

Dealing with missing values in the O_Size parameter ... warning ... they are not "NA's"

```
# Unfortunately, there is another kind of missing value ... A missing "blank"
# O_size parameter
head(mart)

# Let's see what those look like.  Their sales look remarkably like the "Small"
# category
ggplot(mart, aes(x=O_Size, y=Sales)) + geom_boxplot()

# Looks like they are scattered among Supermarket type 1's and Grocery stores
# But again the distribution looks remarkably like "Small" stores
ggplot(mart, aes(x=O_Size, y=O_Type)) + geom_jitter()

# One last test, looking at the O_Loc.  This doesn't quite match the small
# pattern.  It looks like the missing values are all in Tier2 and Tier3, and
# All the other tier2-s  are indeed small, but the tier 3's are mostly meduim
# with a bit of high.  Erring on the size of smaller we could justfy the
# substitution
#
#  O_Size = Blank && O_Loc == Tier2 --> Small   (most of them are this way)
#  O_Size = Blank && O_Loc == Tier3 --> Medium
ggplot(mart, aes(x=O_Size, y=O_Loc)) + geom_jitter()

# This code does the more complicated substitution, but if you like you can also
# try the simpler substitute of small for the blanks (as it is most of them)
oldMart = mart
mart$O_Size = replace(mart$O_Size, (mart$O_Size == "") & (mart$O_Loc == "Tier 2"),
"Small")
mart$O_Size = replace(mart$O_Size, (mart$O_Size == "") & (mart$O_Loc == "Tier 3"),
"Medium")
head(mart)

ggplot(mart, aes(x=O_Size, y=O_Loc)) + geom_jitter()
```

Now look at the Visibility variable

```
ggplot(mart, aes(x=Type, y=Visibility, fill=Type)) + geom_boxplot()

# Is it reasonable that some of these have a visibility of 0?  That would mean
# perhaps that they aren't on display at all ... but yet they have non-zero sales?

# The following density plot shows a large number of zero visibilities.  The
# density plot doesn't fall off on the left.
ggplot(mart, aes(x=Visibility)) + geom_density()

# One option here would be to treat these 0's as missing. Which is what we
# will do here for the sake of showing another tehcnique for filling missing
# values.  The usual technique would be to substitute the mean (or median)
# which leaves the mean/median unchange but can DRASTICALLY reduce variance
# and can also cause very strange residual plots (with a large number of identica
# values :)
#
# So, the R "mice" package has a method that will use the overall distribution of
# the rest of the values relative to the other variables in the set to fill these
# in such a way that the overall distribution will remain the same
library(mice)

martTmp = mart
martTmp$Visibility = replace(martTmp$Visibility, martTmp$Visibility == 0, NA)
ggplot(martTmp, aes(x=Visibility)) + geom_density()

md.pattern(martTmp)

# This will take a long time to run
tempData <- mice(martTmp,m=5,maxit=7,meth='pmm',seed=500)
summary(tempData)
```

Now let's look at the correlation matrix

```
head(mart)

# Several categorical variables are ordinal, so we change them to numeric
# types for this analysis
levels(mart$O_Size)
levels(mart$O_Loc)
levels(mart$O_Type)

martNumeric = mart[, c(12, 2, 4, 6, 8, 9, 10, 11)]
martNumeric$O_Estab = as.numeric(martNumeric$O_Estab)
martNumeric$O_Loc = as.numeric(martNumeric$O_Loc)
martNumeric$O_Size = as.numeric(martNumeric$O_Size)
martNumeric$O_Type = as.numeric(martNumeric$O_Type)

martNumericClean = martNumeric[complete.cases(martNumeric), ]

# This will fail on any field that has missing values!
c = cor(martNumericClean)
mean(mart$Weight)
corrplot(c, method="ellipse")
```

```
# This finds the following major correlations
#
#     Sales <---> MSRP
#     O_Size <---> O_Loc ... but this is a converted ordinal
#
# Let's take a look at Sales and MSRP

# A worrisome number of outliers, both in the plot vs Type and
vs Outlet
# (Notice the two grocery store sales in the sales vs Outlet plot
:)
ggplot(mart, aes(x=Type, y=Sales, fill=Type)) + geom_boxplot()
+ coord_flip()
ggplot(mart, aes(x=Outlet, y=Sales, fill=Outlet)) +
geom_boxplot() + coord_flip()

# Fairly stable
ggplot(mart, aes(x=Type, y=MSRP, fill=Type)) +
geom_boxplot() + coord_flip()

# We get a distinct linear relationship!  Obviously because
Sales = #Units * MSRP!
ggplot(mart, aes(x=MSRP, y=Sales, fill=Type)) + geom_point()
```

Dealing with the Sales/MSRP correlation

```
# The big problem is the Salse vs MSRP correlation because MSRP is obviously
# highly corrleated with Sales simply because Sales = #Units * MSRP (unless
# there is a sale, but that will be a % of MSRP).  So let's look at what happens
# when we divide Sales by MSRP to better normalize that value.
ggplot(mart, aes(x=Type, y=Sales/MSRP, fill=Type)) + geom_boxplot() + coord_flip()
ggplot(mart, aes(x=Outlet, y=Sales/MSRP, fill=Outlet)) + geom_boxplot() + coord_flip()

# Since this gives fewer outliers, we will go ahead and roll MSRP into sales
# and create a new parameter of interest
#
#    stdSales = Sales / MSRP
#
# Note that if we predict stdSales, we will be able to predict Sales by computing
#
#    predSales = predStdSales * MSRP
#
# So this is a valid transform.
mart$stdSales = mart$Sales / mart$MSRP
```

Creating a couple of categories

```
# We had a curious pattern in MSRP in one of the previous graphs
ggplot(mart, aes(x=MSRP, y=Sales)) + geom_point()

# Notice that the MSRP's tend to clump into four distinct groups.  We
# reduce the "bandwidth" of the density calculation to better see the drops
# You can reduce the bandwidth until the top of the curve starts to become
# chaotic.  We get the minimums here by inspection.
#
# Note ... this is one time when you want many tick marks on the x-axis :)
ggplot(mart, aes(x=MSRP)) + geom_density(bw=2) + scale_x_continuous(breaks=(0:30) *
10) +
   geom_vline(xintercept=70, color="red") + geom_vline(xintercept=136, color="red") +
   geom_vline(xintercept=204, color="red")

# So, let's create a new field for the price category, it will be
#
#    0 < MSRP <= 70  --> "Low"
#   70 < MSRP <= 136 --> "Medium"
#  136 < MSRP <= 204 --> "High"
#  204 < MSRP        --> "VeryHigh"
#
# We start by setting the whole field to "Low" and then we will reset the
# others
mart$PriceCat = factor(rep("Low", nrow(mart)), levels=c("Low", "Medium", "High",
"VeryHigh"))
levels(mart$PriceCat)

# Note that we can do this without compound tests (i.e. (MSRP > 70) && (MSRP < 136))
# Since we will over-write the higher valued categories afterwords.  Note that this
# is HIGHLY dependent on the order that these are executed!
mart$PriceCat[mart$MSRP > 70] = "Medium"
mart$PriceCat[mart$MSRP > 136] = "High"
mart$PriceCat[mart$MSRP > 204] = "VeryHigh"

head(mart)
```

Dealing with many categorical levels

```
# Type has 16 levels which can be rather a problem for regression analysis
# as it would mean 15 dummies!
levels(mart$Type)

# Luckily we get a gift here (which you may not get in other sets, but there
# may be other ways to get the same information :) The product "ID" has a
# leading two-letter code that tells what broad product category the product
# belongs to ... in fact threre are only three of them!
#
#   FD = Food
#   DR = Drinks
#   NC = Non-consumables
unique(substring(mart$ID, 1, 2))

# So let's make a new category "ProductCat"
mart$ProductCat = substring(mart$ID, 1, 2)
head(mart)

# Now, let's look at the other datapoints with respect to this new category.
# Normalized by MSRP this gives a very even set of distributions ... oh well
ggplot(data=mart, aes(x=ProductCat, y=stdSales, fill=ProductCat)) +
geom_boxplot()
```

Now create some initial analyses

```
library(ggplot2)
library(GGally)
library(car)

names(mart)

# Reorder the fields to get the new Y at the first column
# and then drop a few (like the old "Sales" field, and the
# "ID" field)
newMart = mart[, c(13, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 14, 15)]
newMart = newMart[complete.cases(newMart), ]

head(newMart)
ggpairs(newMart[, -5])  # Type has too many levels for GGally to process

# Don't start off with this! All the categoricals make it incomprehensible
# and nothing will likely be important!  It does tell us that there is
# about R_squared = .42 out there to capture, but it doesn't tell us HOW!
fit = lm(stdSales ~ ., data=newMart)
summary(fit)

names(newMart)
```

Suddenly the type of supermarket becomes key. Remember, all of these dummies are compared to a baseline of "Grocery". This of course, makes sense but it doesn't really tell us anything about improving sales for a specific type of market except to "Get bigger" ... sigh

```
# Try one with just some numericals?
fit1 = lm(stdSales ~ Weight + Visibility + O_Estab,
data=newMart)
summary(fit1)

# How about some of the categoricals ... not much here
fit2 = lm(stdSales ~ ProductCat + PriceCat, data=newMart)
summary(fit2)

library(leaps)
fitFull = lm(stdSales ~ ., data=newMart)
fitNull = lm(stdSales ~ 1, data=newMart)
stepFit = step(fitNull, scope = list(upper=fitFull),
data=newMart, direction="both")
summary(stepFit)

# Looks like most of the difference is coming from the outlet ...
but not from the outletSize?
# More invesitgation is needed. So, let's take out "Outlet" and
see what we get
fitFull = lm(stdSales ~ . - Outlet, data=newMart)
fitNull = lm(stdSales ~ 1, data=newMart)
stepFit = step(fitNull, scope = list(upper=fitFull),
data=newMart, direction="both")
summary(stepFit)
```

Exploring PCA

```
# For this, we will treat the ordinals as numerical, excluding the priceCategory
# because that one is precisely based on another numerical variable that we
have

martNumeric = mart[, c(13, 2, 4, 6, 8, 9, 10, 11)]
head(martNumeric)
martNumeric$O_Estab = as.numeric(martNumeric$O_Estab)
martNumeric$O_Loc = as.numeric(martNumeric$O_Loc)
martNumeric$O_Size = as.numeric(martNumeric$O_Size)
martNumeric$O_Type = as.numeric(martNumeric$O_Type)

martNumericClean = martNumeric[complete.cases(martNumeric), ]

head(martNumeric)
pMart = prcomp(martNumericClean[2:8], scale=T)
print(pMart)
summary(pMart)

biplot(pMart)

library(psych)
pMartRot = principal(martNumericClean[2:8], nfactors=3, covar=F)
summary(pMartRot)
print(pMartRot)

source("PCA_Plot.R")
PCA_Plot(pMart)
PCA_Plot_Psyc(pMartRot)

factanal(martNumericClean[2:8], factors=3)
```

PCA & PCA Plot

Cars

PCA_Plot functions

```
PCA_Plot = function(pcaData)
{
  library(ggplot2)

  theta = seq(0,2*pi,length.out = 100)
  circle = data.frame(x = cos(theta), y =
sin(theta))
  p = ggplot(circle,aes(x,y)) + geom_path()

  loadings = data.frame(pcaData$rotation,
.names = row.names(pcaData$rotation))
  p + geom_text(data=loadings,
mapping=aes(x = PC1, y = PC2, label =
.names, colour = .names,
fontface="bold")) +
    coord_fixed(ratio=1) + labs(x = "PC1", y
= "PC2")
}
```

```
PCA_Plot_Secondary = function(pcaData)
{
  library(ggplot2)

  theta = seq(0,2*pi,length.out = 100)
  circle = data.frame(x = cos(theta), y =
sin(theta))
  p = ggplot(circle,aes(x,y)) + geom_path()

  loadings = data.frame(pcaData$rotation,
.names = row.names(pcaData$rotation))
  p + geom_text(data=loadings,
mapping=aes(x = PC3, y = PC4, label =
.names, colour = .names,
fontface="bold")) +
    coord_fixed(ratio=1) + labs(x = "PC3", y
= "PC4")
}
```

```
PCA_Plot_Psyc = function(pcaData)
{
  library(ggplot2)

  theta = seq(0,2*pi,length.out = 100)
  circle = data.frame(x = cos(theta), y = sin(theta))
  p = ggplot(circle,aes(x,y)) + geom_path()

  loadings = as.data.frame(unclass(pcaData$loadings))
  s = rep(0, ncol(loadings))
  for (i in 1:ncol(loadings))
  {
    s[i] = 0
    for (j in 1:nrow(loadings))
      s[i] = s[i] + loadings[j, i]^2
    s[i] = sqrt(s[i])
  }

  for (i in 1:ncol(loadings))
    loadings[, i] = loadings[, i] / s[i]

  loadings$.names = row.names(loadings)

  p + geom_text(data=loadings, mapping=aes(x = PC1, y = PC2, label =
.names, colour = .names, fontface="bold")) +
    coord_fixed(ratio=1) + labs(x = "PC1", y = "PC2")
}
```

```
PCA_Plot_Psyc_Secondary = function(pcaData)
{
  library(ggplot2)

  theta = seq(0,2*pi,length.out = 100)
  circle = data.frame(x = cos(theta), y = sin(theta))
  p = ggplot(circle,aes(x,y)) + geom_path()

  loadings = as.data.frame(unclass(pcaData$loadings))
  s = rep(0, ncol(loadings))
  for (i in 1:ncol(loadings))
  {
    s[i] = 0
    for (j in 1:nrow(loadings))
      s[i] = s[i] + loadings[j, i]^2
    s[i] = sqrt(s[i])
  }

  for (i in 1:ncol(loadings))
    loadings[, i] = loadings[, i] / s[i]

  loadings$.names = row.names(loadings)

  print(loadings)
  p + geom_text(data=loadings, mapping=aes(x = PC3, y = PC4, label =
.names, colour = .names, fontface="bold")) +
    coord_fixed(ratio=1) + labs(x = "PC3", y = "PC4")
}
```

Analyze the cars dataset

```
# Read in the cars dataset
cars04 = read.csv("cars-fixed04.dat")
carsNumeric = cars04[, 8:18]

head(carsNumeric)

# Check the correlation plot
c = cor(carsNumeric)
corrplot(c, method="ellipse", order="AOE")

# Looks like two primary components, but the second one
# may be subdivided into more than one, since dealer and
# retail are essentially perfectly correlated and engine
# and cylinders are highly correlated
carsPCA = prcomp(carsNumeric)
summary(carsPCA)
round(carsPCA$rotation, 2)
plot(carsPCA)
PCA_Plot(carsPCA)

# One component is giving us everything and it is an even
# mixture of Retail and Dealer.  This makes sense because
# Those values being the price and dealer cost are very large
# Numbers.  So we re-scale and re-run PCA
carsPCA = prcomp(carsNumeric, scale=T)
summary(carsPCA)
round(carsPCA$rotation, 2)
plot(carsPCA)
PCA_Plot(carsPCA)
```

```
# Note that the first component is showing MPG positive and
# everything else negative.  So, the first component is saying
# that the MPG are negatively related to all the performance
# characteristics of the car

# For the second PC, it has positive contributions from the
# Wheelbase, length, width and height, I.E. the size of the
# car, but then negative correlations of horsepower and
# costs.  To understand this one better let's look at how some
# of the cars "score" on this scale.  The scores are in the
# prcomp's $x parameter
head(carsPCA$x[, 1:2])

# NSX = hybrid sports car
# RL = Luxury sedan
# MDX = SUV

# So this parameter separates larger cars (sedans, SUV's,
minivans)
# from smaller sports cars,
```

Canonical Correlation Manual Case

Iris

Manual canonical correlation computation

```
data(iris)
head(iris)
irisNum = iris[, 1:4]
X = iris[, 1:2]
Y = iris[, 3:4]

head(irisNum)

totalCor = cor(irisNum)
round(totalCor, 2)
Rxx = totalCor[1:2, 1:2]
Ryy = totalCor[3:4, 3:4]
Rxy = totalCor[1:2, 3:4]
Ryx = totalCor[3:4, 1:2]
RxxInv = solve(Rxx)
RyyInv = solve(Ryy)

R = RyyInv %*% Ryx %*% RxxInv %*% Rxy
e = eigen(R)
cCor1 = sqrt(e$values)
round(cCor1, 2)
```

```
totalCov = cov(irisNum)
Cxx = totalCov[1:2, 1:2]
Cyy = totalCov[3:4, 3:4]
Cxy = totalCov[1:2, 3:4]
Cyx = totalCov[3:4, 1:2]
CxxInv = solve(Cxx)
CyyInv = solve(Cyy)

Cy = CyyInv %*% Cyx %*% CxxInv %*% Cxy
e = eigen(Cy)
cCor2 = sqrt(e$values)
round(cCor2, 2)

Cx = CxxInv %*% Cxy %*% CyyInv %*% Cyx
e = eigen(Cx)
cCor3 = sqrt(e$values)
round(cCor3, 2) totalCov = cov(irisNum)
Cxx = totalCov[1:2, 1:2]
Cyy = totalCov[3:4, 3:4]
Cxy = totalCov[1:2, 3:4]
Cyx = totalCov[3:4, 1:2]
CxxInv = solve(Cxx)
CyyInv = solve(Cyy)

Cy = CyyInv %*% Cyx %*% CxxInv %*% Cxy
e = eigen(Cy)
cCor2 = sqrt(e$values)
round(cCor2, 2)

Cx = CxxInv %*% Cxy %*% CyyInv %*% Cyx
e = eigen(Cx)
cCor3 = sqrt(e$values)
round(cCor3, 2)
```

Now, for the eigenvectors. These depend on which of the matrices above we choose

```
e = eigen(C1)
vY = e$vectors

e = eigen(C2)
vX = e$vectors

c = cc(X, Y)

# These differ only by a constant in each column
c$xcoef
vX

c$ycoef
vY

# The following all give scales of each other
c1 = cancor(X, Y)
c2 = cc(X, Y)
c3 = cca(X, Y)
c1$xcoef / c2$xcoef
c2$xcoef / c3$xcoef
c2

helio.plot(c3, cv=1, x.name="Sepal Values",
        y.name="Petal Values", type="variance")

helio.plot(c3, cv=2, x.name="Sepal Values",
        y.name="Petal Values", type="variance")
```

Canonical Correlation Auto Case

Iris

Exploring correlations between sepal and petal

```
data(iris)
head(iris)
write.table(iris, file="iris.txt", sep=" ", row.names=F)
round(cor(iris[, 1:4]), 2)

sL = iris$Sepal.Length
sW = iris$Sepal.Width
pL = iris$Petal.Length
pW = iris$Petal.Width

cor(sL, pL)
cor(sL, pW)
cor(.5 * sL + .5 * sW, pL)
# Try some others!

# Now, try this one
cor(sL - sW, pL - pW)

plot(sL - sW, pL - pW)
plot(sL - sW, pL - pW, col=iris[, 5])
# A very odd plot.  These points seem to be falling on a grid when
# we subtract :)

# Finally try this one
cor(sL - .85*sW, .86*pL - .7*pW)

plot(sL - .85*sW, .86*pL - .7*pW)
plot(sL - .85*sW, .86*pL - .7*pW, col=iris[, 5])
```

This is a nice function for computing the Wilks lambdas for CCA data from the CCA library's method. It computes the wilkes lambas the degrees of freedom and the p-values

```
ccaWilks = function(set1, set2, cca)
{
 ev = ((1 - cca$cor^2))
 ev

 n = dim(set1)[1]
 p = length(set1)
 q = length(set2)
 k = min(p, q)
 m = n - 3/2 - (p + q)/2
 m

 w = rev(cumprod(rev(ev)))

 # initialize
 d1 = d2 = f = vector("numeric", k)

 for (i in 1:k)
 {
  s = sqrt((p^2 * q^2 - 4)/(p^2 + q^2 - 5))
  si = 1/s
  d1[i] = p * q
  d2[i] = m * s - p * q/2 + 1
  r = (1 - w[i]^si)/w[i]^si
  f[i] = r * d2[i]/d1[i]
  p = p - 1
  q = q - 1
 }

 pv = pf(f, d1, d2, lower.tail = FALSE)
 dmat = cbind(WilksL = w, F = f, df1 = d1, df2 = d2, p = pv)
}
```

```
sepal = iris[, 1:2]
petal = iris[, 3:4]

# This gives us the cannonical correlates, but no significance tests
c = cancor(sepal, petal)
c

# The CCA library has more extensive functionality
library(CCA)
matcor(sepal, petal)

ccIris = cc(sepal, petal)
ccIris$cor

ls(ccIris)
ccIris$xcoef
ccIris$ycoef

loadingsIris = comput(sepal, petal, ccIris)
ls(loadingsIris)
loadingsIris$corr.X.xscores
loadingsIris$corr.Y.yscores

wilksIris = ccaWilks(sepal, petal, ccIris)
round(wilksIris, 2)

# Now, let's calcualte the standardized coefficients
s1 = diag(sqrt(diag(cov(sepal))))
s1 %*% ccIris$xcoef

s2 = diag(sqrt(diag(cov(petal))))
s2 %*% ccIris$ycoef

# A basic visualization of the cannonical correlation
plt.cc(ccIris)
```

try it with yacca

```
library(yacca)
c2 = cca(sepal, petal)
c2

helio.plot(c3, cv=1, x.name="Sepal Values",
        y.name="Petal Values")

helio.plot(c3, cv=2, x.name="Sepal Values",
        y.name="Petal Values")

ls(c2)

# Perform a chisquare test on C2
c2
ls(c2)
c2$chisq
c2$df
summary(c2)
round(pchisq(c2$chisq, c2$df, lower.tail=F), 3)
```

Linear Discriminant Analysis

```
library(MASS)
head(iris)
result = lda(Species ~ Sepal.Length + Sepal.Width + Petal.Length + Petal.Width, data=iris )
result

plot(result)
```

Canonical Correlation Case

Sales

This is a nice function for computing the Wilks lambdas for CCA data from the CCA library's method

It computes the wilkes lambas the degrees of freedom and te p-values

```
ccaWilks = function(set1, set2, cca)
{
 ev = ((1 - cca$cor^2))
 ev

 n = dim(set1)[1]
 p = length(set1)
 q = length(set2)
 k = min(p, q)
 m = n - 3/2 - (p + q)/2
 m

 w = rev(cumprod(rev(ev)))

 # initialize
 d1 = d2 = f = vector("numeric", k)

 for (i in 1:k)
 {
  s = sqrt((p^2 * q^2 - 4)/(p^2 + q^2 - 5))
  si = 1/s
  d1[i] = p * q
  d2[i] = m * s - p * q/2 + 1
  r = (1 - w[i]^si)/w[i]^si
  f[i] = r * d2[i]/d1[i]
  p = p - 1
  q = q - 1
 }

 pv = pf(f, d1, d2, lower.tail = FALSE)
 dmat = cbind(WilksL = w, F = f, df1 = d1, df2 = d2, p = pv)
}
```

Reading the data

```
ds = read.table("sales.txt", header=F)
names(ds) = c("Growth", "Profit", "NewAcct", "Creativity", "MechReas", "AbstReas", "Math")
head(ds)
```

Separating the data. We will use employee test scores to try to predict their sales performance

```
sales = ds[, 1:3]
scores = ds[, 4:7]

head(sales)
head(scores)
```

```
# The CCA library has more extensive functionality
library(CCA)

# First invesitgate the combined correlation matrix, and test the
# cross correlations.  To do this we use the "matcor" function which
# computes correlation matrices between two datasets
c = matcor(scores, sales)

# Then we pull out the upper right block of correlations that compare
# the sales and scores variables
cCross = c$XYcor[1:3, 4:7]
round(cCross, 2)

# Now, run a correlation test
library(psych)
p = corr.p(cCross, nrow(scores))
p

# Next, we run the canonlical correlation
ccSales = cc(scores, sales)
ccSales$cor
```

```
# First, let's test the model for significance.  We are quite sure that
# We will find some correlation here because our matrix above showed
# Significance, but the Wilks test confirms this
wilksSales = ccaWilks(scores, sales, ccSales)
round(wilksSales, 2)


# To understand the canonical correlates, we look at the raw coefficients
# These can be interpreted exactly like components in PCA except that we
# have two sets of them.  Notice, however, that it is the RELATIVE
# contributions of each variable that are important, not the absolute
# size of the contribution
names(ccSales)
round(-ccSales$xcoef, 3)  # Since the first column is negative, we negate these
round(-ccSales$ycoef, 3)  # Notice that this makes no difference in the relationship between them
```

```
# To help us better understand the components, we look at the correlations
# between each of the variates and the original variables that make them up.
# This creates two "loadings" matrices very much like the loadings matrix
# of PCA
#
# You will notice that this gives a different picture of the relationship between the x's and
# their variates.  Math was a lower "coefficient" but has the highest correlation with the
# variate.  This means in a real sense that math contributes most highly to this variate.
# The reason for this difference is in the size of the contributions and in how the other
variables
# also contribute.  If Math swings more, it might contribute more even though its
coefficient is
# smaller
loadingsSales = comput(scores, sales, ccSales)
ls(loadingsSales)
loadingsSales$corr.X.xscores
loadingsSales$corr.Y.yscores

# Let's plot the first two variates against each other.  We do this by looking at the scores.
plot(loadingsSales$xscores[,1], loadingsSales$yscores[,1])
cor(loadingsSales$xscores[,1], loadingsSales$yscores[,1])

# Last we look at the relationship between the variables and the correlates from the other
dataset
# This gives us a better view of how the variables from one set relate to the other correlate
and
# how we might predict one from the other
loadingsSales$corr.X.yscores   # How do the y-variates depend on the x-variables.  Most
important for prediction
loadingsSales$corr.Y.xscores

# A basic visualization of the cannonical correlation
plt.cc(ccSales)
```

try it with yacca

```
library(yacca)
c2 = cca(sepal, petal)
c2

helio.plot(c3, cv=1, x.name="Sepal Values",
        y.name="Petal Values")

helio.plot(c3, cv=2, x.name="Sepal Values",
        y.name="Petal Values")

ls(c2)

# Perform a chisquare test on C2
c2
ls(c2)
c2$chisq
c2$df
summary(c2)
round(pchisq(c2$chisq, c2$df, lower.tail=F), 3)
```

Correspondence Analysis Case

Hair Eye Color

This dataset is a 3D table with two "pages”

One for men and one for women

```
head(HairEyeColor)

Men = HairEyeColor[,,1]
Women = HairEyeColor[,,2]

Men
Women

library(ca)

prop.table(Men, 1)  # Row Percentages
prop.table(Men, 2)  # Col Percentages

fit = ca(Men)
summary(fit)
fit

plot(fit)
Men

plot(fit, mass=T, contrib="absolute",
    map="rowgreen", arrows=c(F, T))
```

Now lets look at women

```
prop.table(Women, 1)  # Row Percentages
prop.table(Women, 2)  # Col Percentages

fit = ca(Women)
summary(fit)
fit

plot(fit)
Women

plot(fit, mass=T, contrib="absolute",
    map="rowgreen", arrows=c(F, T))

library(vcd)
mosaic(Men, shade=TRUE, legend=TRUE)
```

Correspondence Analysis Case

Letter Frequencies

```
library(ca)

lFreq = read.table("LetterFrequencies.txt", header=T)
lFreq

# Make a new field with the initials
firstInitial = substr(lFreq$FName, 1, 1)
lastInitial = substr(lFreq$LName, 1, 1)
Initials = paste(firstInitial, lastInitial, lFreq$TextID, sep="")

# Now remove the first three fields and set the row names to
be the Initials
lFreq = lFreq[, c(4:19)]

rownames(lFreq) = Initials
lFreq

write.table(lFreq, "letterFreq.txt", sep="\t")

# Compute the correspondence matrix
P = lFreq / sum(lFreq)
round(P, 3)
```

```
# To get the chiSquared statistics for each, we need the row and column sums
rSum = rowSums(P)
rSum
cSum = colSums(P)
cSum

# Then we compute all the products as a new matrix
mu = as.matrix(rSum) %*% t(as.matrix(cSum))
mu

# Finally we make a matrix of the deviations and sum them
tmp = (P - mu)^2 / mu
chiSquared = sum(lFreq) * sum(tmp)
chiSquared

# Compute the probability of getting a chiSquared that high
# Note the degrees of freedom here
pchisq(chiSquared, (nrow(lFreq) - 1) * (ncol(lFreq) - 1), lower.tail=F)

# The ca library has a nice correspondence analysis function
c = ca(lFreq)
c$N
c$rowcoord

summary(c)
plot(c)

eigenvals = get_eigenvalue(c)

# This plot puts arrows to the letters so that we can compare
# Their relative frequencies to the texts
plot(c, mass=T, contrib="absolute",
    map="rowgreen", arrows=c(F, T))
```