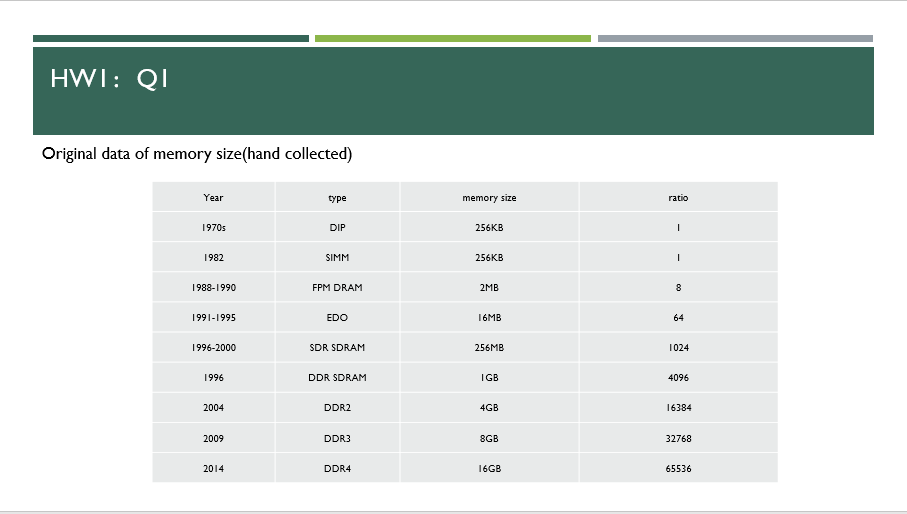
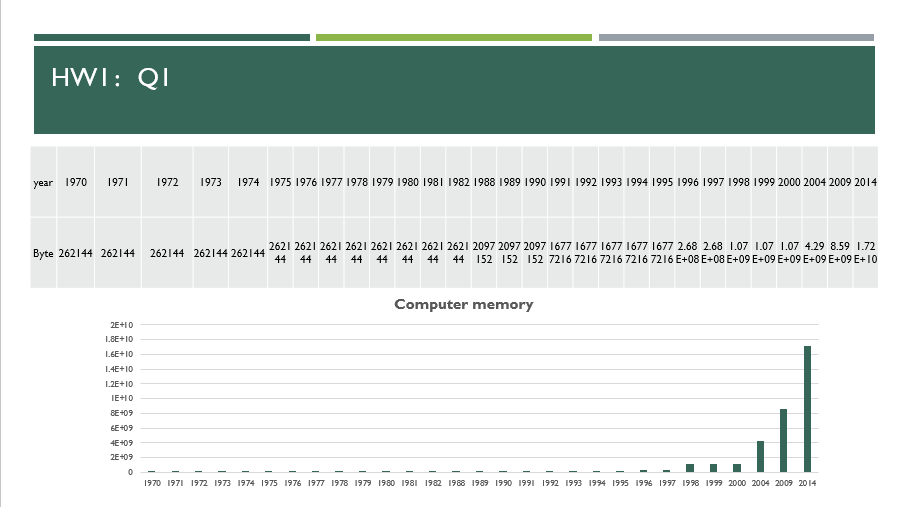
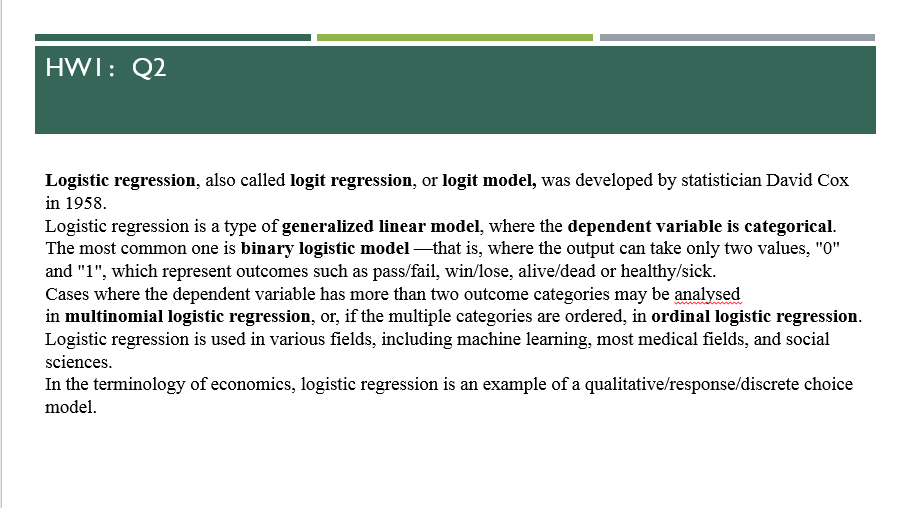
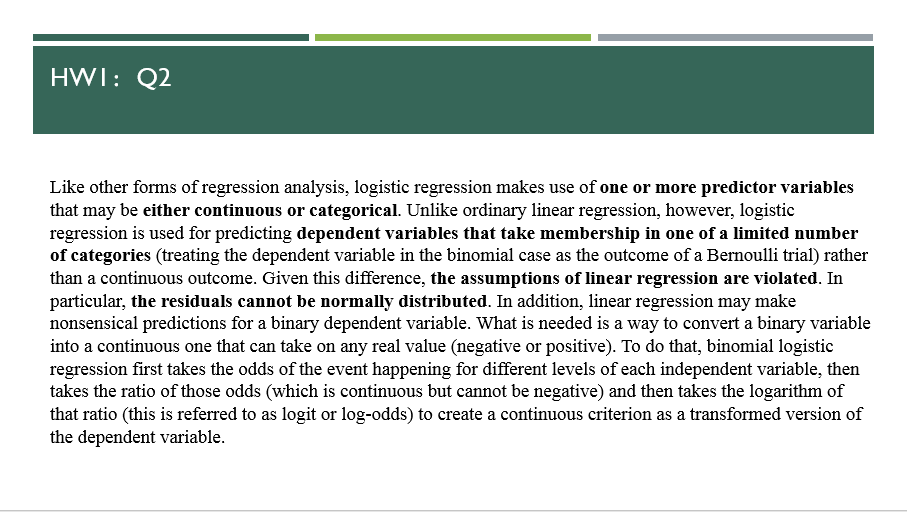
**HW1:**



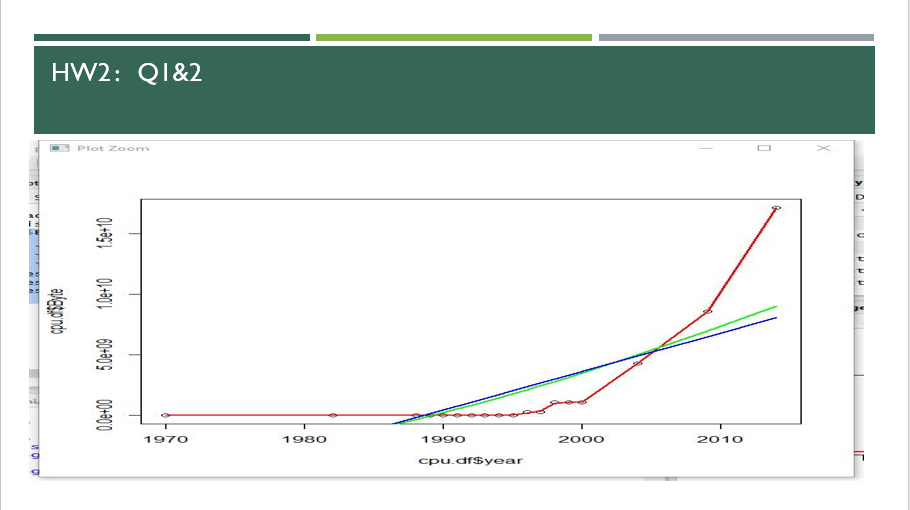


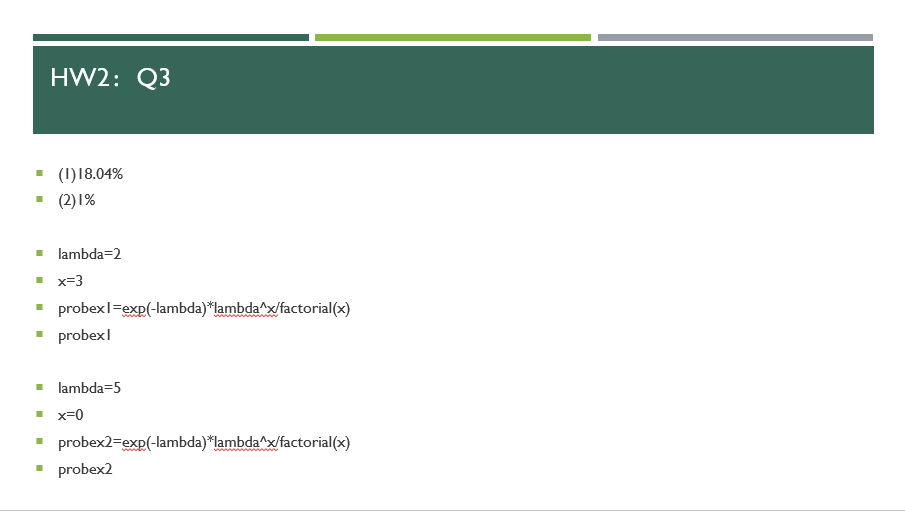




**HW2:**







# EX1&EX2

cpu.df = read.csv("byte.csv",header = TRUE)

plot(cpu.df$Byte~cpu.df$year,title(main = "The development of computer memory",cex.main= 0.8))

splines.reg.l1 = smooth.spline(x = cpu.df$year, y = cpu.df$Byte, spar = 0.2)

splines.reg.l2 = smooth.spline(x = cpu.df$year, y = cpu.df$Byte, spar = 1)

splines.reg.l3= smooth.spline(x = cpu.df$year, y = cpu.df$Byte, spar = 2)

lines(splines.reg.l1, col = "red", lwd = 2) # regression line with lambda = 0.2

lines(splines.reg.l2, col = "green", lwd = 2) # regression line with lambda = 1

lines(splines.reg.l3, col = "blue", lwd = 2) # regression line with lambda = 2

# EX3

lambda=2

x=3

probex1=exp(-lambda)\*lambda^x/factorial(x)

probex1

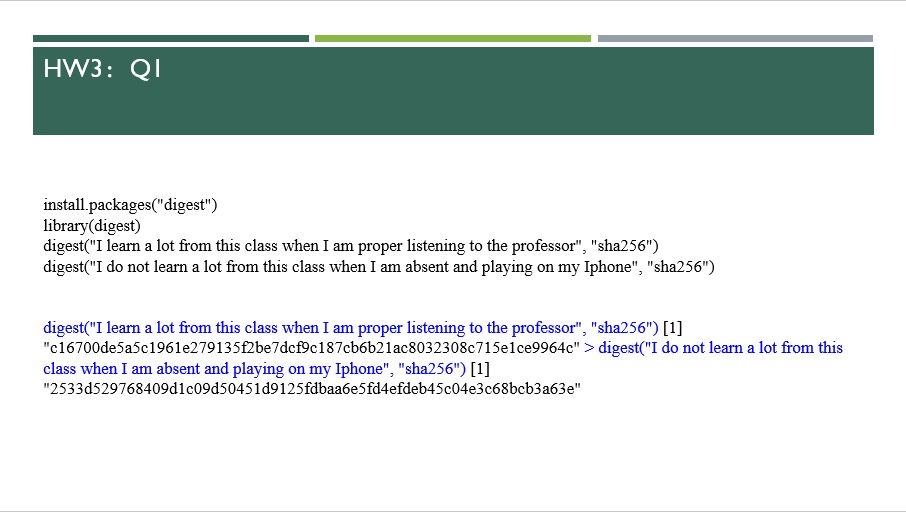
lambda=5

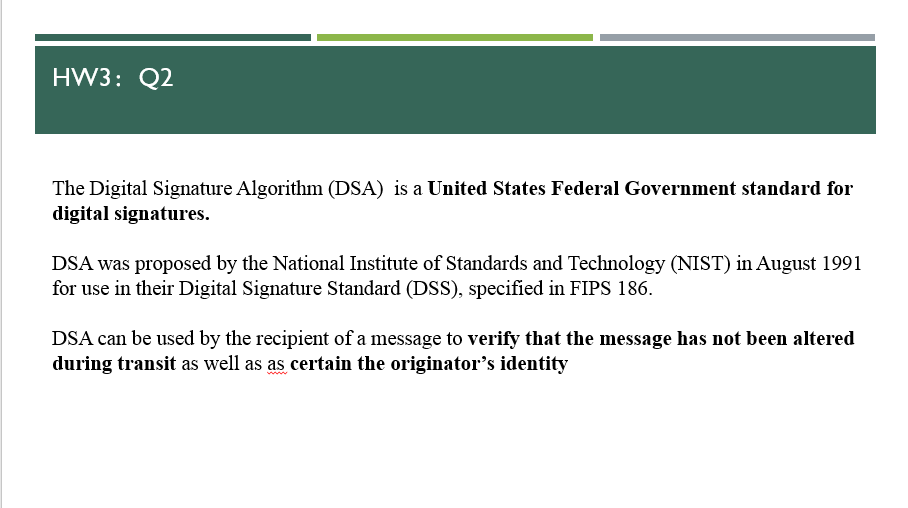
x=0

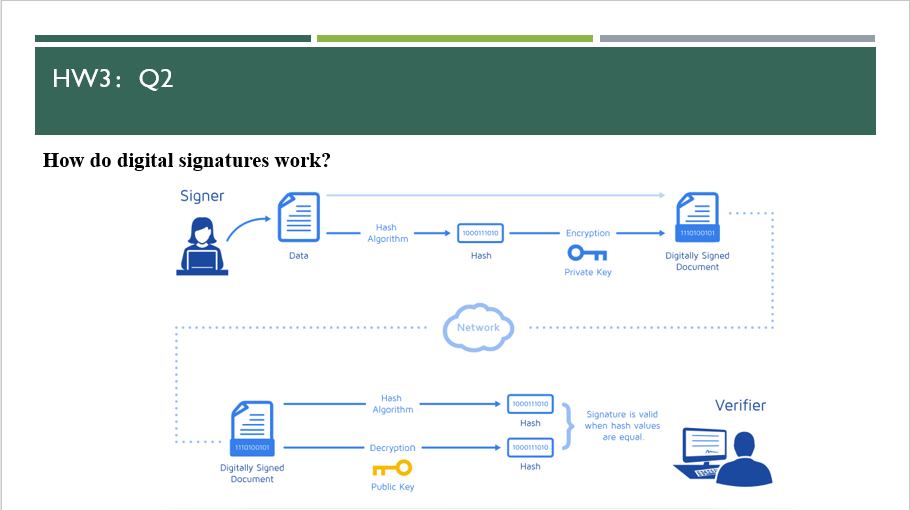
probex2=exp(-lambda)\*lambda^x/factorial(x)

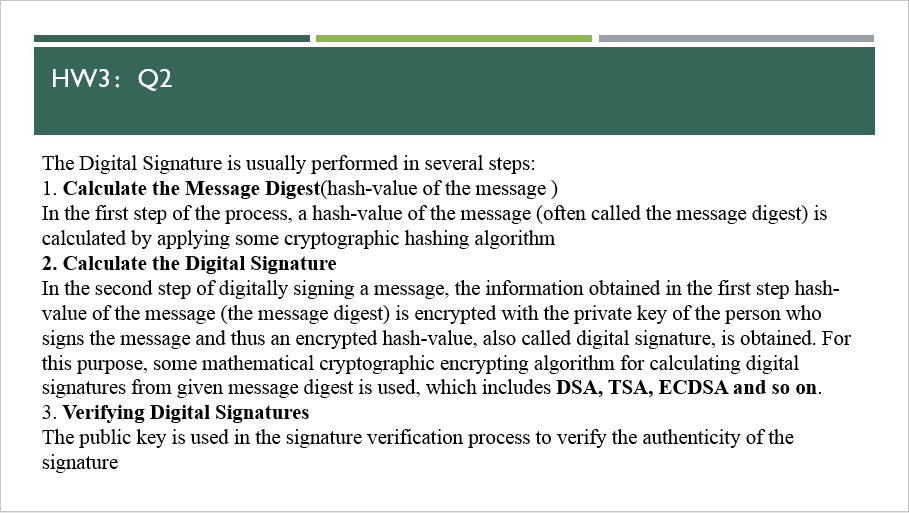
probex2

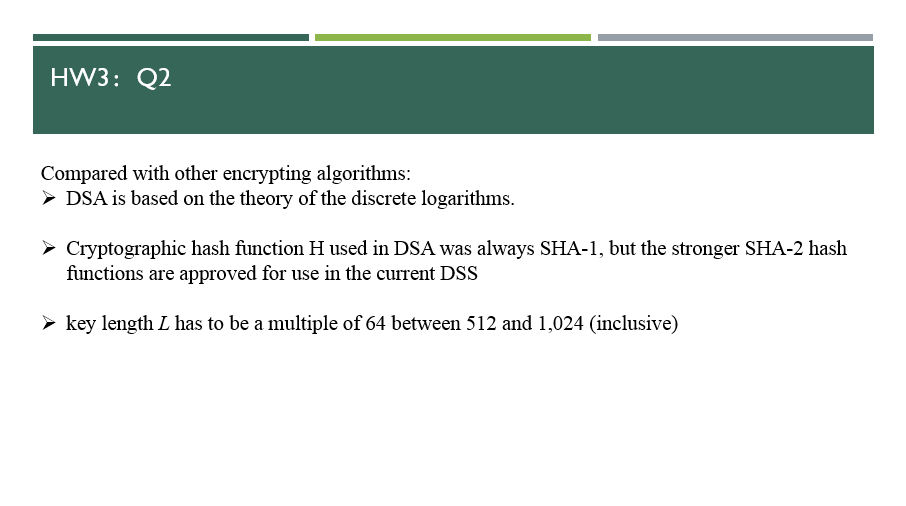
**HW3:**

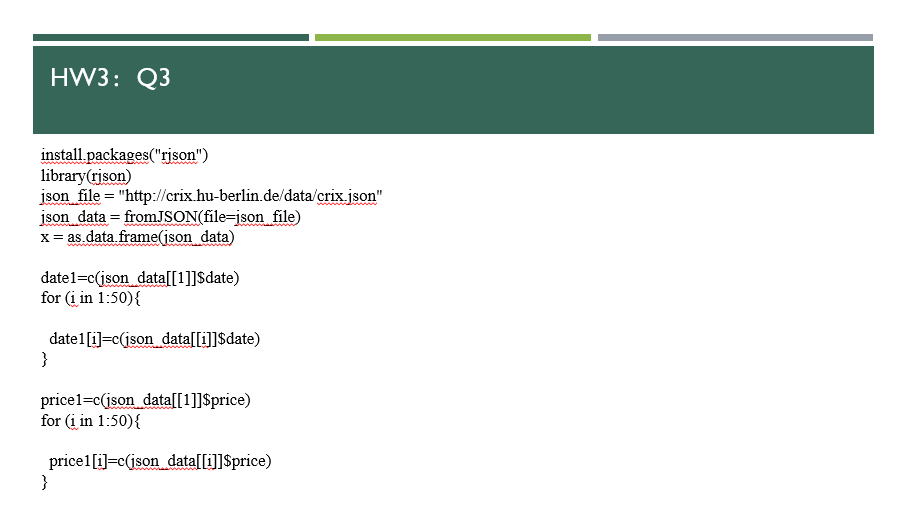


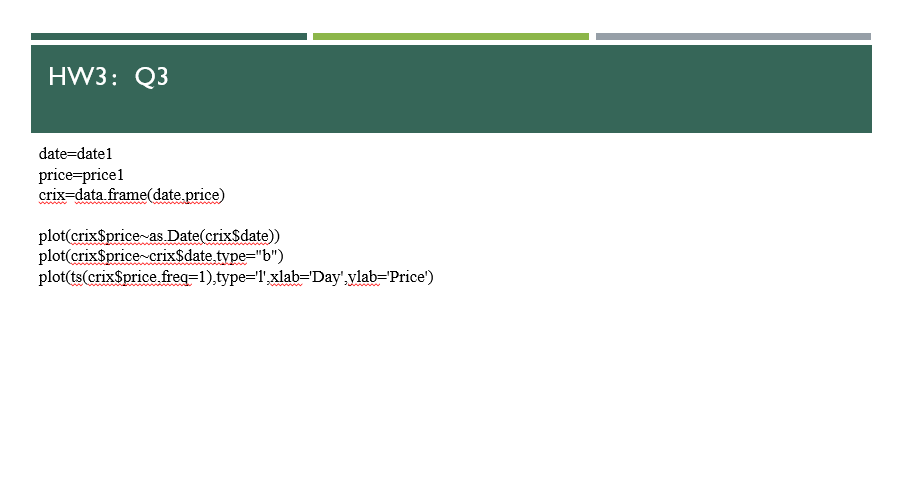


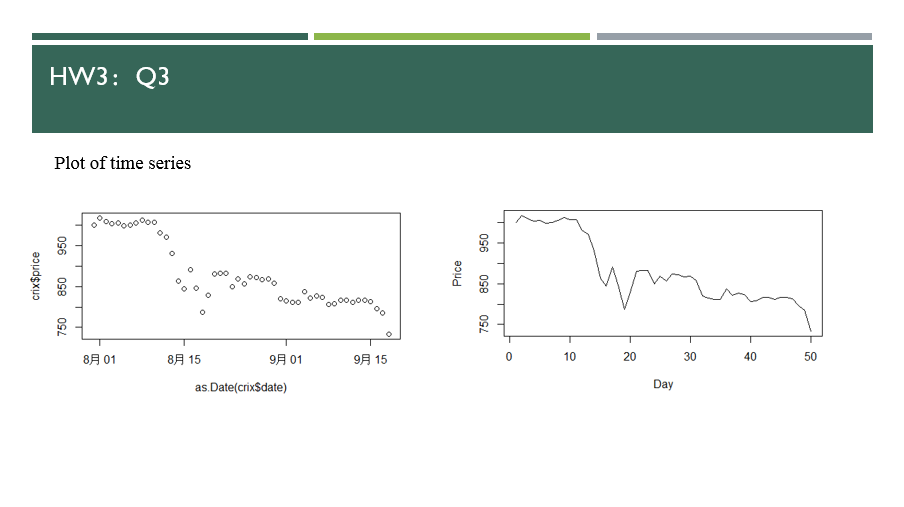


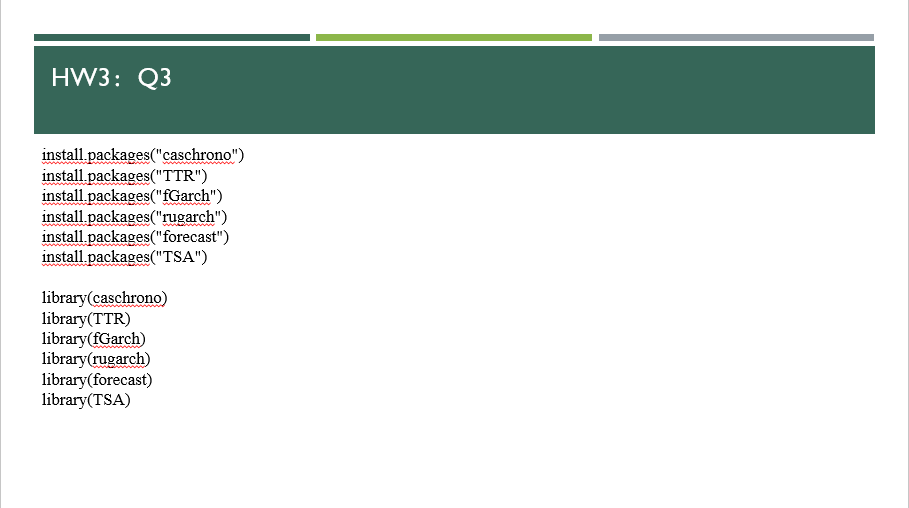


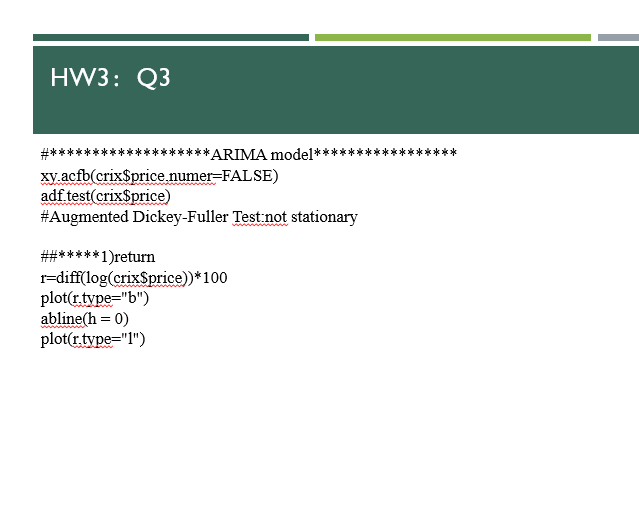


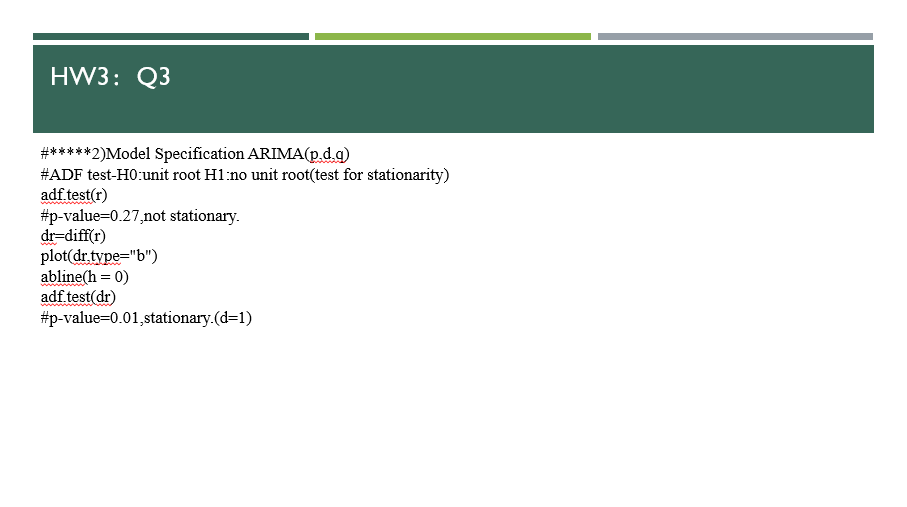


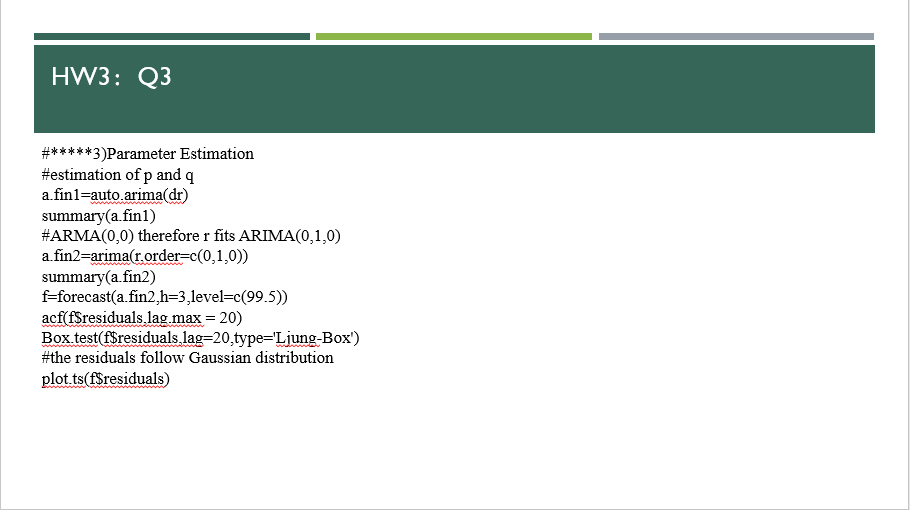


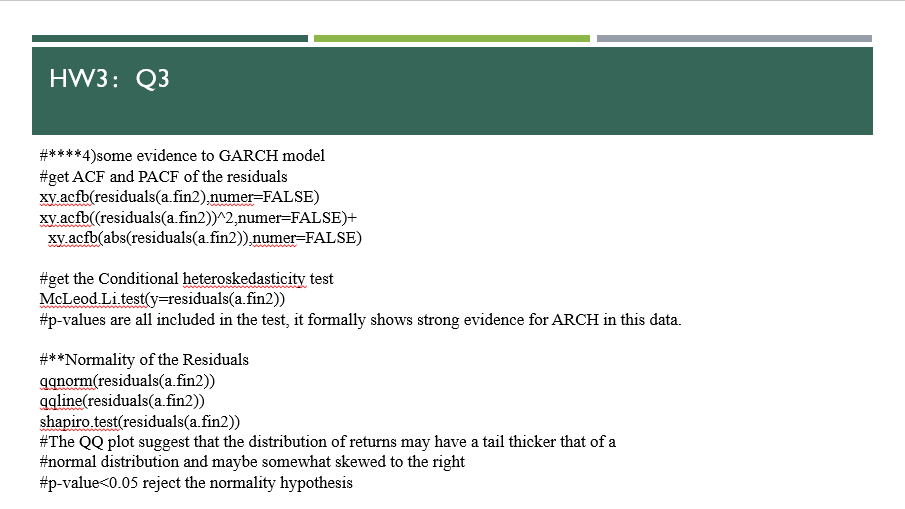


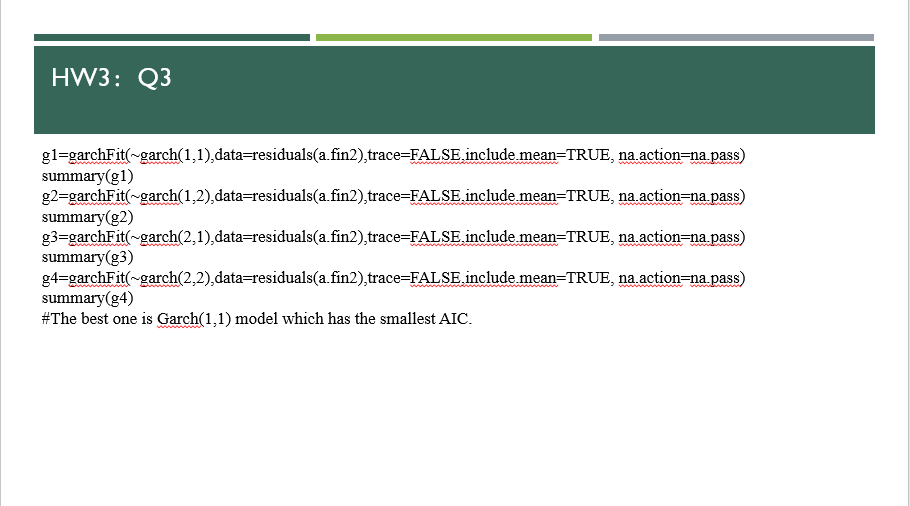












#Q1

library(digest)

digest("I learn a lot from this class when I am proper listening to the professor","sha256")

digest("I do not learn a lot from this class when I am absent and playing on my Iphone","sha256")

#Q3&4

library(rjson)

json\_file = "http://crix.hu-berlin.de/data/crix.json"

json\_data = fromJSON(file=json\_file)

x = as.data.frame(json\_data)

date1=c(json\_data[[1]]$date)

for (i in 1:2354){

date1[i]=c(json\_data[[i]]$date)

}

price1=c(json\_data[[1]]$price)

for (i in 1:2354){

price1[i]=c(json\_data[[i]]$price)

}

date=date1

price=price1

crix=data.frame(date,price)

plot(crix$price~as.Date(crix$date))

plot(crix$price~crix$date,type="b")

plot(ts(crix$price,freq=1),type='l',xlab='Day',ylab='Price')

library(caschrono)

library(TTR)

library(fGarch)

library(rugarch)

library(forecast)

library(TSA)

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*ARIMA medel\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

xy.acfb(crix$price,numer=FALSE)

adf.test(crix$price)

#Augmented Dickey-Fuller Test:not stationary

##\*\*\*\*\*1)return

r=diff(log(crix$price))\*100

plot(r,type="b")

abline(h = 0)

plot(r,type="l")

#\*\*\*\*\*2)Model Specification ARIMA(p,d,q)

#ADF test-H0:unit root H1:no unit root(test for stationarity)

adf.test(r)

#p-value=0.27,not stationary.

dr=diff(r)

plot(dr,type="b")

abline(h = 0)

adf.test(dr)

#p-value=0.01,stationary.(d=1)

#\*\*\*\*\*3)Parameter Estimation

#estimation of p and q

a.fin1=auto.arima(dr)

summary(a.fin1)

#ARMA(0,0) therefore r fits ARIMA(0,1,0)

a.fin2=arima(r,order=c(0,1,0))

summary(a.fin2)

help("forecast.Arima")

f=forecast(a.fin2,h=3,level=c(99.5))

acf(f$residuals,lag.max = 20)

Box.test(f$residuals,lag=20,type='Ljung-Box')

#the residuals follow Gaussian distribution

plot.ts(f$residuals)

#\*\*\*\*4)some evidence to GARCH model

#get ACF and PACF of the residuals

xy.acfb(residuals(a.fin2),numer=FALSE)

xy.acfb((residuals(a.fin2))^2,numer=FALSE)+

xy.acfb(abs(residuals(a.fin2)),numer=FALSE)

#get the Conditional heteroskedasticity test

McLeod.Li.test(y=residuals(a.fin2))

#p-values are all included in the test, it formally shows strong evidence for ARCH in this data.

#\*\*Normality of the Residuals

qqnorm(residuals(a.fin2))

qqline(residuals(a.fin2))

shapiro.test(residuals(a.fin2))

#The QQ plot suggest that the distribution of returns may have a tail thicker that of a

#normal distribution and maybe somewhat skewed to the right

#p-value<0.05 reject the normality hypothesis

g1=garchFit(~garch(1,1),data=residuals(a.fin2),trace=FALSE,include.mean=TRUE, na.action=na.pass)

summary(g1)

g2=garchFit(~garch(1,2),data=residuals(a.fin2),trace=FALSE,include.mean=TRUE, na.action=na.pass)

summary(g2)

g3=garchFit(~garch(2,1),data=residuals(a.fin2),trace=FALSE,include.mean=TRUE, na.action=na.pass)

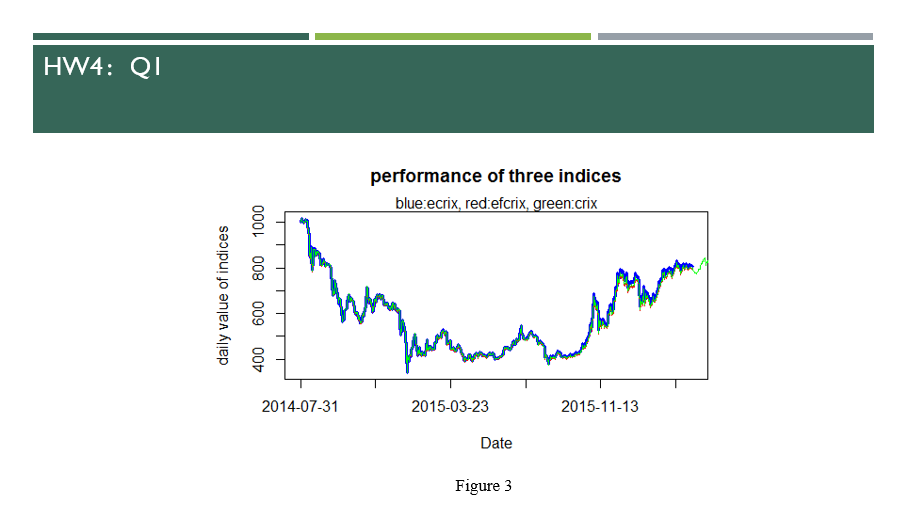
summary(g3)

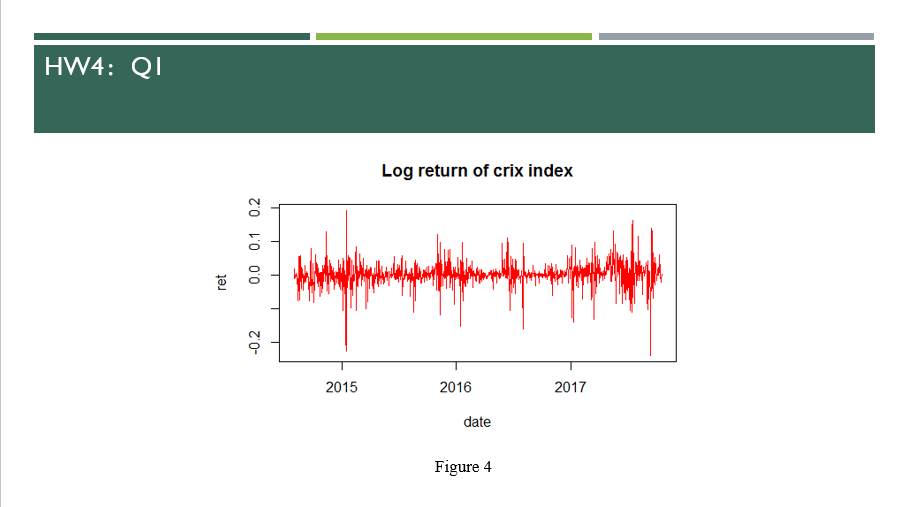
g4=garchFit(~garch(2,2),data=residuals(a.fin2),trace=FALSE,include.mean=TRUE, na.action=na.pass)

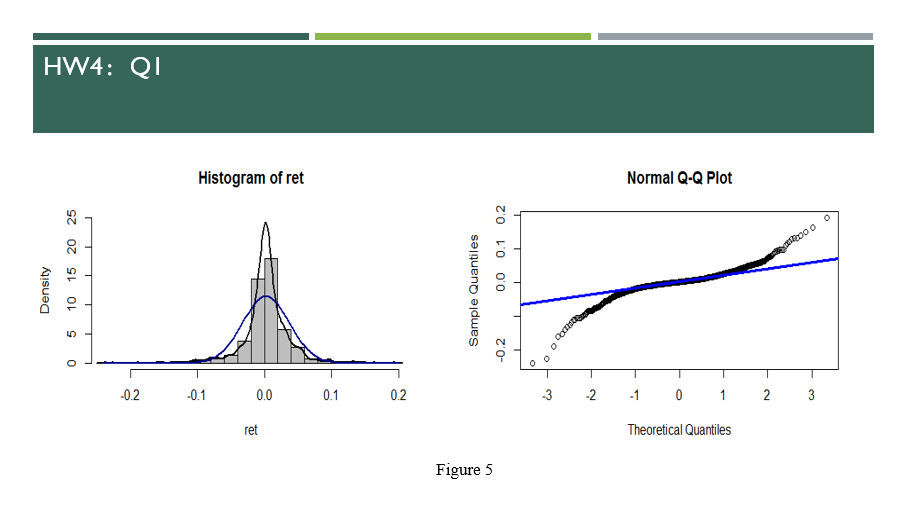
summary(g4)

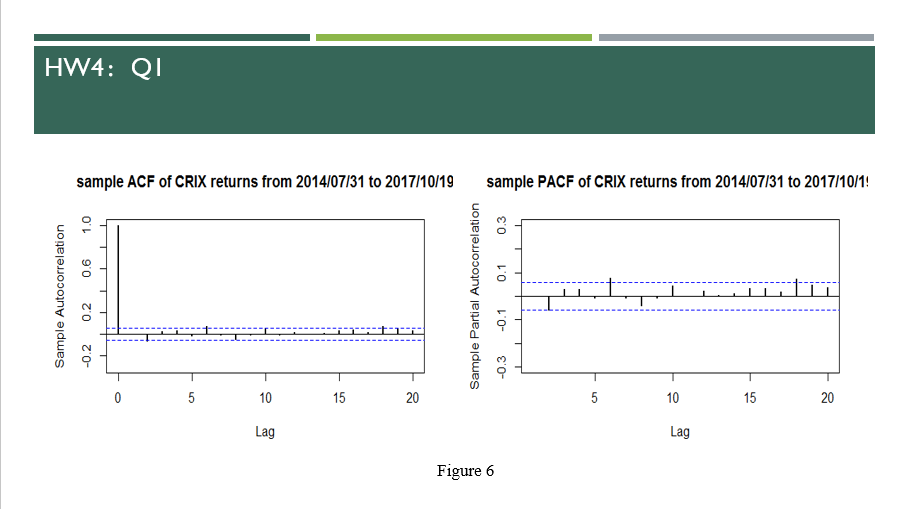
#The best one is Garch(1,1) model which has the smallest AIC.

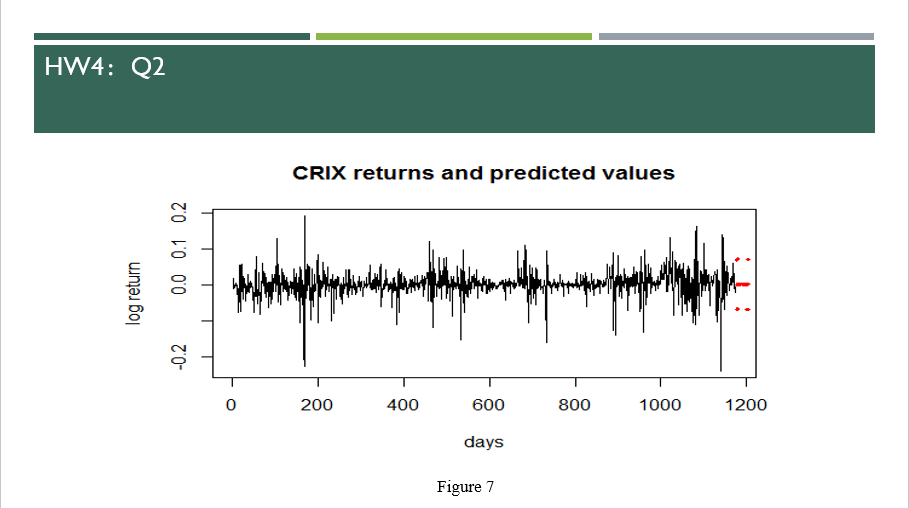
**HW4:**











#figure 3:crix&ecrix&efcrix

setwd("c:/Users/dell/Desktop")

load("ecrix.RData")

load("efcrix.RData")

plot(ecrix, type = "l", col = "blue", xaxt = "n", lwd = 3, main = "performance of three indices", xlab = "Date",

ylab = "daily value of indices")

lines(efcrix, col = "red")

library(rjson)

json\_file = "http://crix.hu-berlin.de/data/crix.json"

json\_data = fromJSON(file=json\_file)

crix\_data\_frame=as.data.frame(json\_data)

x=crix\_data\_frame

dim(x)

n=dim(x)

a=seq(1,n[2],2)

b=seq(2,n[2],2)

data=t(x[1,a])

price=t(x[1,b])

ts.plot(price)

plot(price)

lines(price, col = "green")

s=seq(1,n[2],n[2]/20)

axis(1, at = s, label = names(ecrix)[s])

mtext("blue:ecrix, red:efcrix，green:crix")

#figure4

library(rjson)

json\_file = "http://crix.hu-berlin.de/data/crix.json"

json\_data = fromJSON(file=json\_file)

x = as.data.frame(json\_data)

date1=c(json\_data[[1]]$date)

for (i in 1:2348){

date1[i]=c(json\_data[[i]]$date)

}

price1=c(json\_data[[1]]$price)

for (i in 1:2348){

price1[i]=c(json\_data[[i]]$price)

}

date=date1

price=price1

crix=data.frame(date,price)

date2=date[-1]

ret=diff(log(price))

plot(ret~as.Date(date2),type="l",col="red",xlab="date",ylab="ret", main="Log return of crix index")

#figure5

mean(ret)

var(ret)

sd(ret)

hist(ret, col = "grey", breaks = 20, freq = FALSE, ylim = c(0, 25), xlab = "ret")

lines(density(ret), lwd = 2)

mu = mean(ret)

sigma = sd(ret)

x = seq(-4, 4, length = 100)

curve(dnorm(x, mean = mean(ret), sd = sd(ret)), add = TRUE, col = "darkblue",

lwd = 2)

qqnorm(ret)

qqline(ret, col = "blue", lwd = 3)

#figure6

libraries = c("zoo", "tseries")

autocorr = acf(ret, lag.max = 20, ylab = "Sample Autocorrelation", main = "sample ACF of CRIX returns from 2014/07/31 to 2017/10/19" ,

lwd = 2, ylim = c(-0.3, 1))

autopcorr = pacf(ret, lag.max = 20, ylab = "Sample Partial Autocorrelation",

main = "sample PACF of CRIX returns from 2014/07/31 to 2017/10/19" , ylim = c(-0.3, 0.3), lwd = 2)

#figure7

# arima model

library(caschrono)

library(TTR)

library(forecast)

library(TSA)

par(mfrow = c(1, 1))

auto.arima(ret)

fit202 = arima(ret, order = c(2, 0, 2))

tsdiag(fit202)

fit202 = arima(ret, order = c(2, 0, 2))

crpre = predict(fit202, n.ahead = 30)

dates = seq(as.Date("31/07/2014", format = "%d/%m/%Y"), by = "days", length = length(ret))

plot(ret, type = "l", ylab = "log return", xlab = "days",

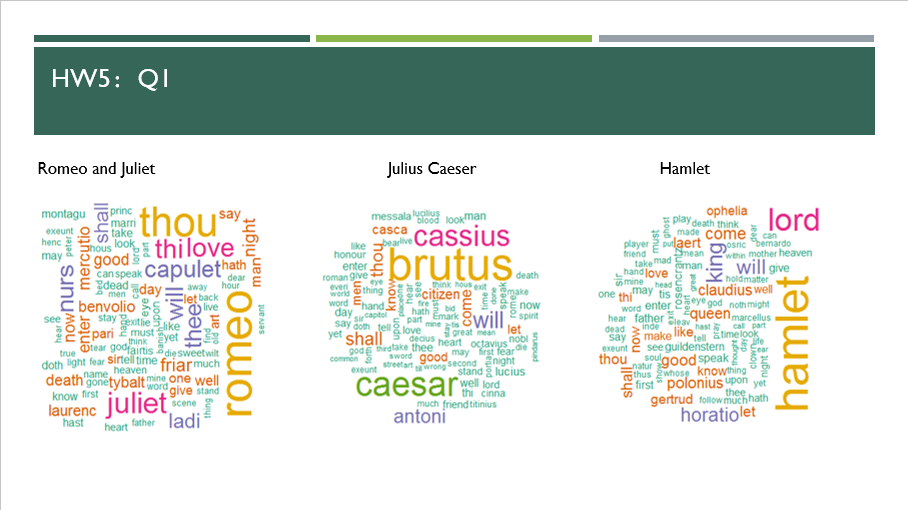
lwd = 1.5, main = "CRIX returns and predicted values")

lines(crpre$pred, col = "red", lwd = 3)

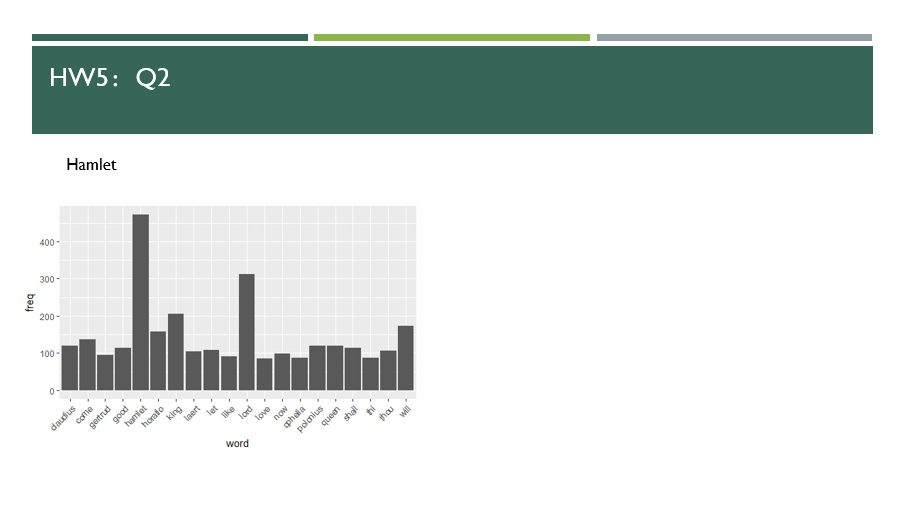
lines(crpre$pred + 2 \* crpre$se, col = "red", lty = 3, lwd = 3)

lines(crpre$pred - 2 \* crpre$se, col = "red", lty = 3, lwd = 3)

**HW5:**







rm(list = ls())

install.packages("RCurl")

install.packages("XML")

library(RCurl)

library(XML)

url1 = "http://shakespeare.mit.edu/romeo\_juliet/full.html"

url2 = "http://shakespeare.mit.edu/julius\_caesar/full.html"

url3 = "http://shakespeare.mit.edu/hamlet/full.html"

html1 = readLines(url1, encoding = "UTF-8")

html2 = readLines(url2, encoding = "UTF-8")

html3 = readLines(url3, encoding = "UTF-8")

html1 = htmlParse(html1, encoding = "UTF-8")

html2 = htmlParse(html2, encoding = "UTF-8")

html3 = htmlParse(html3, encoding = "UTF-8")

#HW5.1 Wordcloud

install.packages("bitops")

install.packages("stringr")

library(bitops)

library(stringr)

abs1 = lapply(url1, FUN = function(x) htmlParse(x, encoding = "Latin-1"))

abs2 = lapply(url2, FUN = function(x) htmlParse(x, encoding = "Latin-1"))

abs3 = lapply(url3, FUN = function(x) htmlParse(x, encoding = "Latin-1"))

clean\_txt = function(x) {

cleantxt = xpathApply(x, "//body//text()

[not(ancestor :: script)][ not(ancestor :: style)]

[not(ancestor :: noscript)] " ,xmlValue)

cleantxt = paste(cleantxt, collapse="\n")

cleantxt = str\_replace\_all(cleantxt, "\n", " ")

cleantxt = str\_replace\_all(cleantxt, "\r", "")

cleantxt = str\_replace\_all(cleantxt, "\t", "")

cleantxt = str\_replace\_all(cleantxt, "<br>", "")

return(cleantxt)

}

cleantxt1 = lapply(abs1,clean\_txt)

cleantxt2 = lapply(abs2,clean\_txt)

cleantxt3 = lapply(abs3,clean\_txt)

vec\_abs1 = unlist(cleantxt1)

vec\_abs2 = unlist(cleantxt2)

vec\_abs3 = unlist(cleantxt3)

###Text Mining

install.packages("tm")

install.packages("SnowballC")

library(tm)

library(SnowballC)

abs1 = Corpus(VectorSource(vec\_abs1))

abs2 = Corpus(VectorSource(vec\_abs2))

abs3 = Corpus(VectorSource(vec\_abs3))

abs\_dtm1 = DocumentTermMatrix(abs1, control = list(

stemming = TRUE, stopwords = TRUE, minWordLength = 3,

removeNumbers = TRUE, removePunctuation = TRUE))

abs\_dtm2 = DocumentTermMatrix(abs2, control = list(

stemming = TRUE, stopwords = TRUE, minWordLength = 3,

removeNumbers = TRUE, removePunctuation = TRUE))

abs\_dtm3 = DocumentTermMatrix(abs3, control = list(

stemming = TRUE, stopwords = TRUE, minWordLength = 3,

removeNumbers = TRUE, removePunctuation = TRUE))