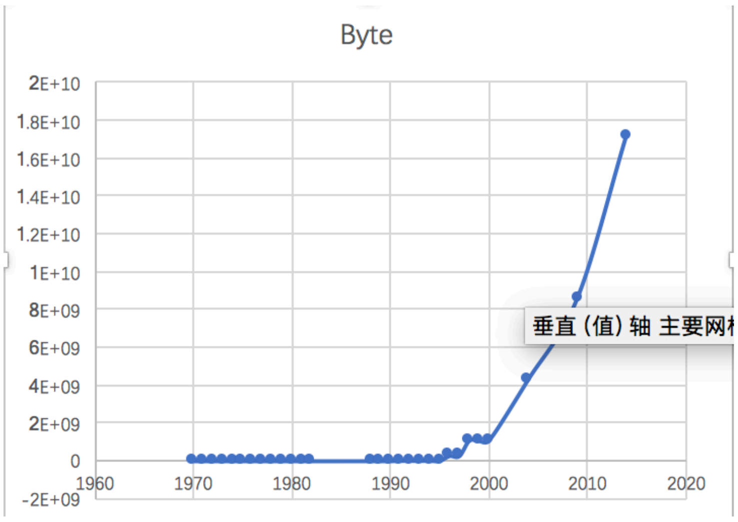
# Homework 1

## 1. memory of PCs

| *year* | *Byte* | *year* | *Byte* |
|---:|---:|---:|---:|
| 1970 | 262144 | 1990 | 2097152 |
| 1971 | 262144 | 1991 | 16777216 |
| 1972 | 262144 | 1992 | 16777216 |
| 1973 | 262144 | 1993 | 16777216 |
| 1974 | 262144 | 1994 | 16777216 |
| 1975 | 262144 | 1995 | 16777216 |
| 1976 | 262144 | 1996 | 268435456 |
| 1977 | 262144 | 1997 | 268435456 |
| 1978 | 262144 | 1998 | 1073741824 |
| 1979 | 262144 | 1999 | 1073741824 |
| 1980 | 262144 | 2000 | 1073741824 |
| 1981 | 262144 | 2004 | 4294967296 |
| 1982 | 262144 | 2009 | 8589934592 |
| 1988 | 2097152 | 2014 | 17179869184 |
| 1989 | 2097152 | | |



## 2. logistic regression

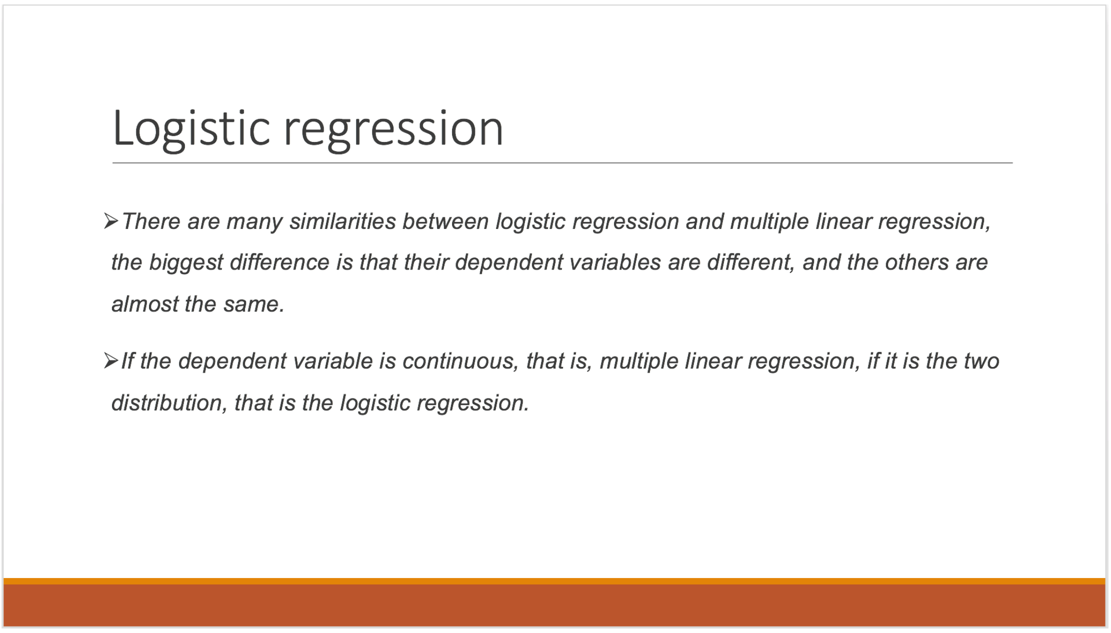
# Logistic regression

27720161153023 PEI MING

## Logistic regression

➢*Logistic regression is a regression model where the dependent variable (DV) is categorical, where a categorical variable is a variable that can take on one of a limited, and usually fixed, number of possible values.*

➢*To explore the risk factors of a disease and predict the probability of a disease according to the risk factors. If we have established the logistic regression model, we can predict the probability of a disease or a certain situation under different independent variables according to the model.*



## Logistic regression

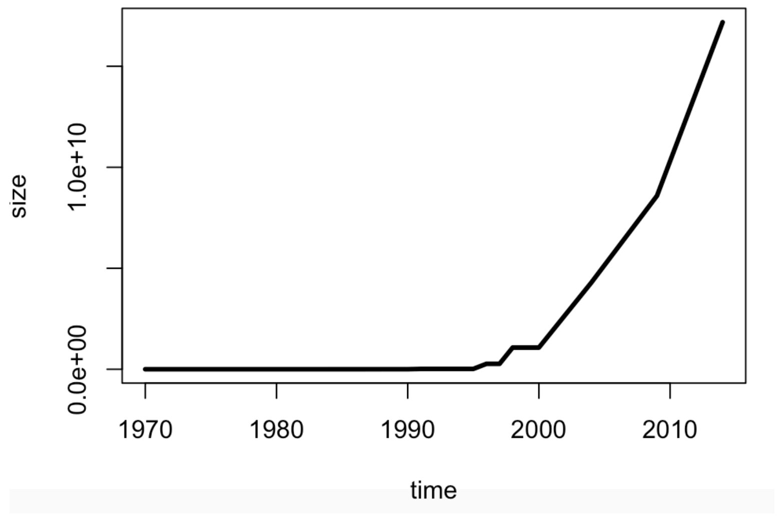
- *There are many similarities between logistic regression and multiple linear regression, the biggest difference is that their dependent variables are different, and the others are almost the same.*
- *If the dependent variable is continuous, that is, multiple linear regression, if it is the two distribution, that is the logistic regression.*

3. **Github Account:** https://github.com/mpmp2013/Home-Work-for-BDIF

## Homework 2

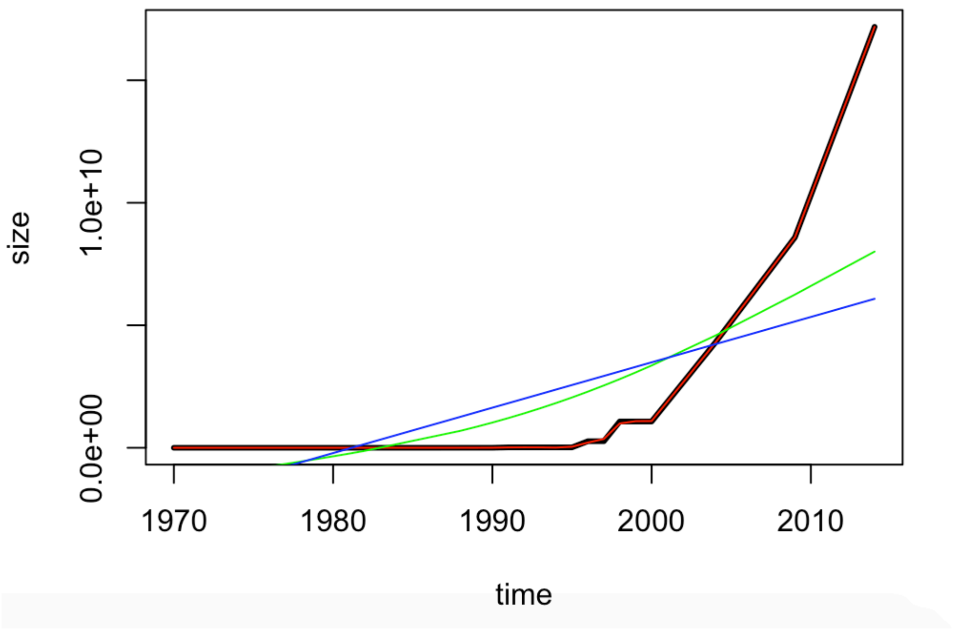
1. **Use R to solve HW #1**

```
library(readr)
RAM_size <- read_csv("~/R data/Home-Work-for-BDIF/RAM_size.csv")
plot(RAM_size,type="l",xlab = "time",ylab = "size",lwd=3)
```



2. **use R with B-spline code to solve HW#1**

```
splines.reg.l1 = smooth.spline(x = RAM_size$year, y = RAM_size$Byte, spar =
0.2)    # lambda = 0.2
splines.reg.l2 = smooth.spline(x = RAM_size$year, y = RAM_size$Byte, spar =
1)    # lambda = 1
splines.reg.l3 = smooth.spline(x = RAM_size$year, y = RAM_size$Byte, spar =
2)    # lambda = 2
lines(splines.reg.l1, col = "red", lwd = 1)    # regression line with lambda = 0.2
lines(splines.reg.l2, col = "green", lwd = 1)    # regression line with lambda = 1
lines(splines.reg.l3, col = "blue", lwd = 1)    # regression line with lambda = 2
```



Comments: The larger the spar is, more smooth the line is.

**3. Poisson Distribution**

```
lambda=4
x=6
dpois(x,lambda)

lambda=5
x=0
dpois(x,lambda)
```

## Homework 3

**1. hash code**

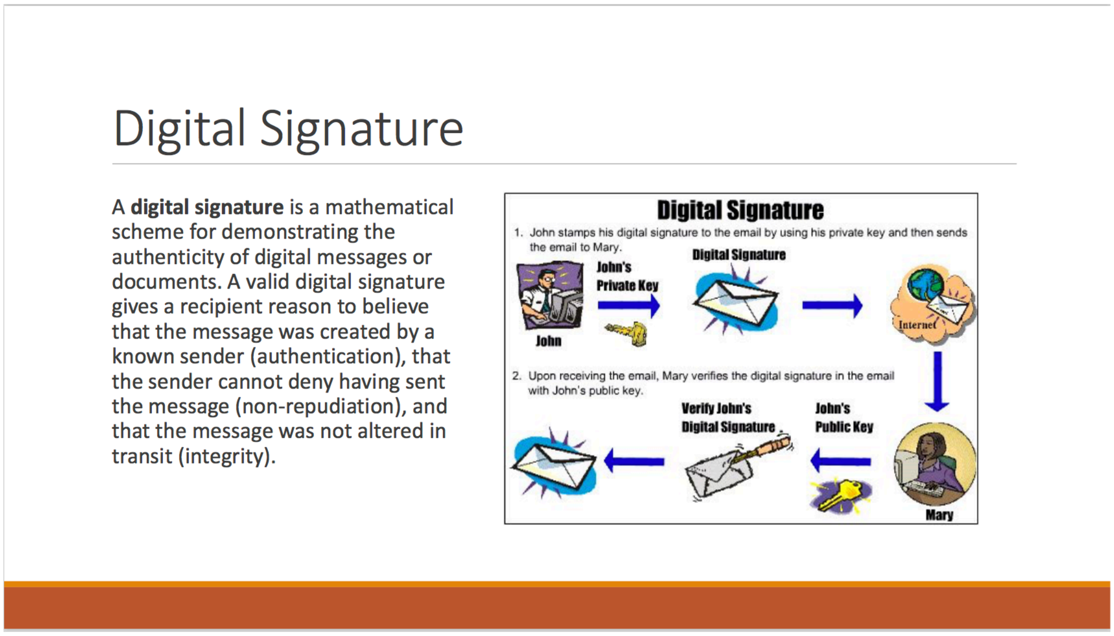
```
#install.packages("digest",repos='http://cran.us.r-projest.org')
```

```
 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")
```

## 2. Digital Signature Algorithms

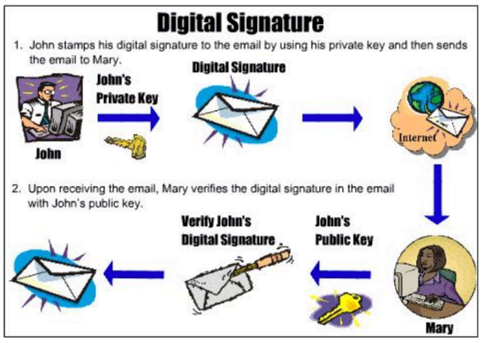
# Digital Signature Algorithm

27720161153023 PEI MING



## Digital Signature

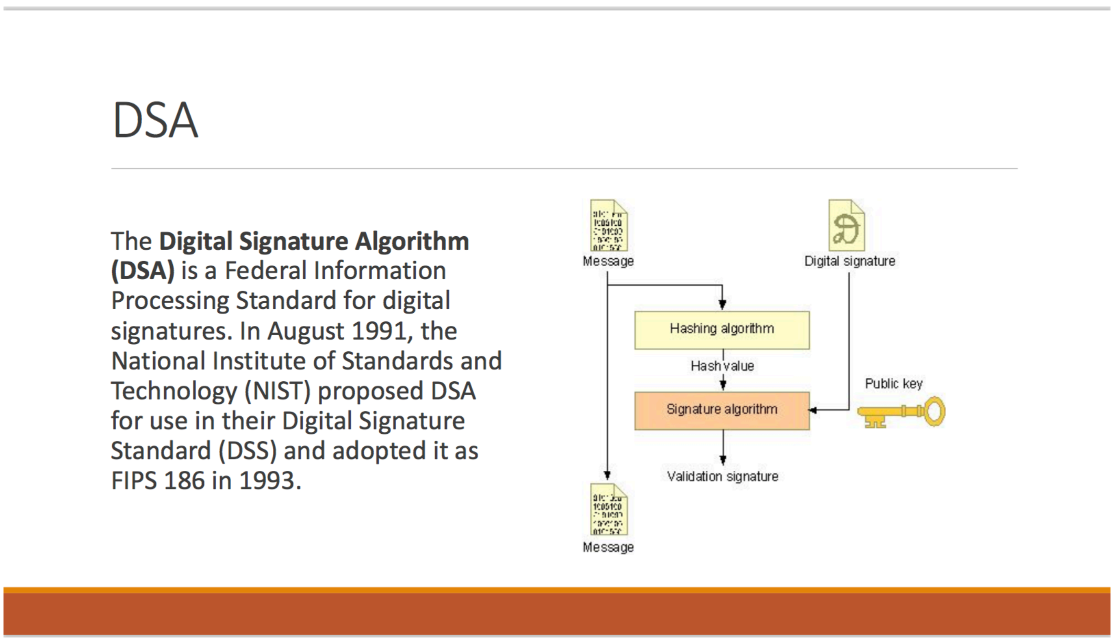
A **digital signature** is a mathematical scheme for demonstrating the authenticity of digital messages or documents. A valid digital signature gives a recipient reason to believe that the message was created by a known sender (authentication), that the sender cannot deny having sent the message (non-repudiation), and that the message was not altered in transit (integrity).



**Digital Signature**

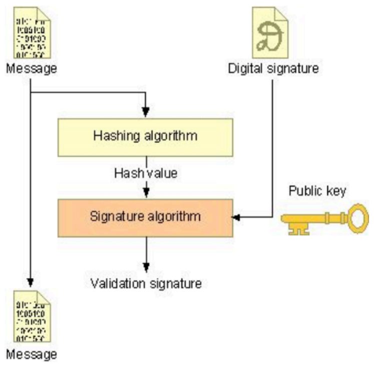
1. John stamps his digital signature to the email by using his private key and then sends the email to Mary.

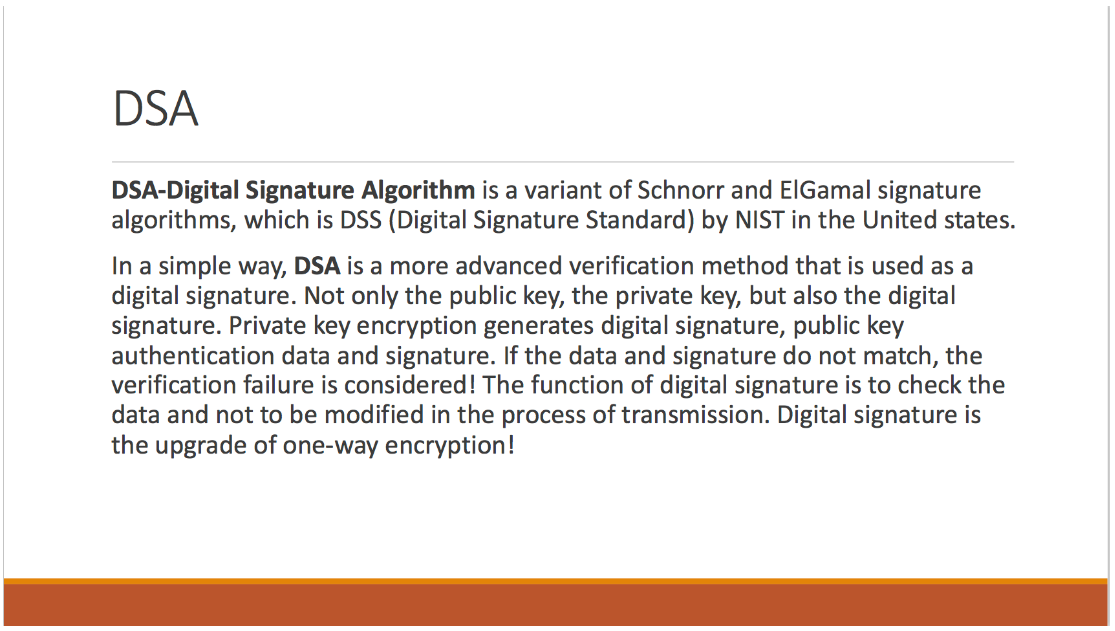
2. Upon receiving the email, Mary verifies the digital signature in the email with John's public key.



## DSA

The **Digital Signature Algorithm (DSA)** is a Federal Information Processing Standard for digital signatures. In August 1991, the National Institute of Standards and Technology (NIST) proposed DSA for use in their Digital Signature Standard (DSS) and adopted it as FIPS 186 in 1993.



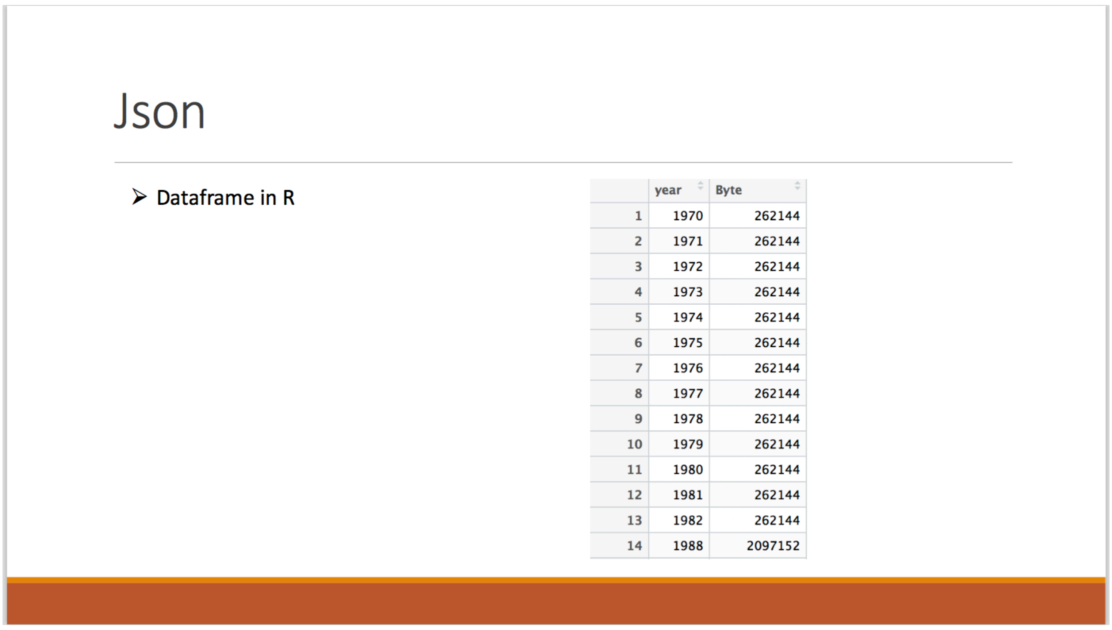


## DSA

**DSA-Digital Signature Algorithm** is a variant of Schnorr and ElGamal signature algorithms, which is DSS (Digital Signature Standard) by NIST in the United states.

In a simple way, **DSA** is a more advanced verification method that is used as a digital signature. Not only the public key, the private key, but also the digital signature. Private key encryption generates digital signature, public key authentication data and signature. If the data and signature do not match, the verification failure is considered! The function of digital signature is to check the data and not to be modified in the process of transmission. Digital signature is the upgrade of one-way encryption!

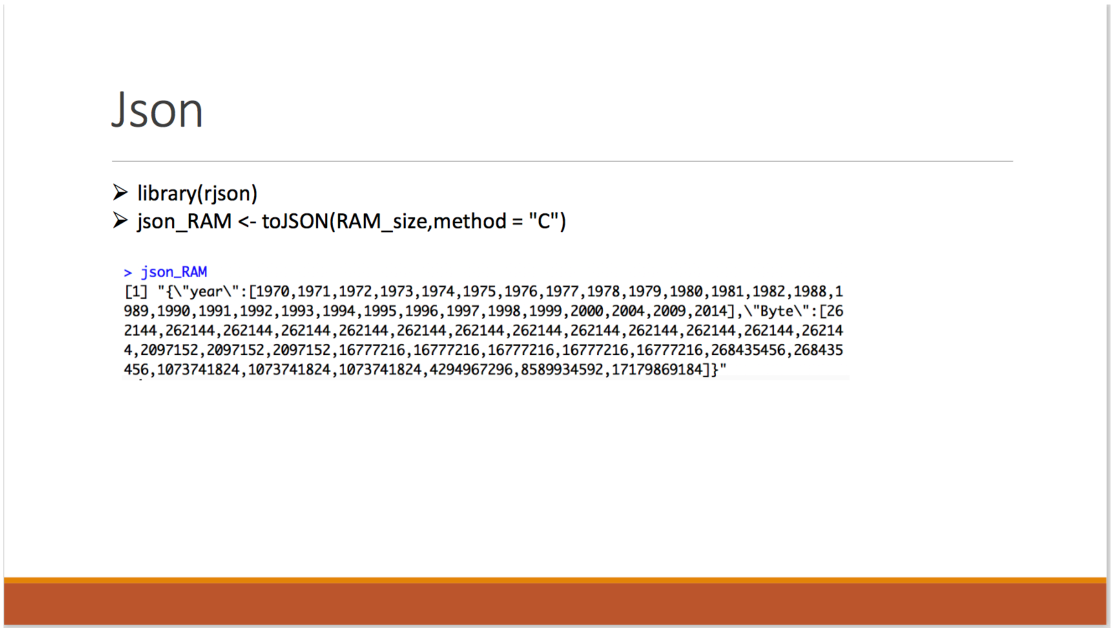
## 3. Json data



## Json

- Dataframe in R

| | year | Byte |
|---|---|---|
| 1 | 1970 | 262144 |
| 2 | 1971 | 262144 |
| 3 | 1972 | 262144 |
| 4 | 1973 | 262144 |
| 5 | 1974 | 262144 |
| 6 | 1975 | 262144 |
| 7 | 1976 | 262144 |
| 8 | 1977 | 262144 |
| 9 | 1978 | 262144 |
| 10 | 1979 | 262144 |
| 11 | 1980 | 262144 |
| 12 | 1981 | 262144 |
| 13 | 1982 | 262144 |
| 14 | 1988 | 2097152 |



## Json

- library(rjson)
- json_RAM <- toJSON(RAM_size,method = "C")

```
> json_RAM
[1] "{\"year\":[1970,1971,1972,1973,1974,1975,1976,1977,1978,1979,1980,1981,1982,1988,1
989,1990,1991,1992,1993,1994,1995,1996,1997,1998,1999,2000,2004,2009,2014],\"Byte\":[26
2144,262144,262144,262144,262144,262144,262144,262144,262144,262144,262144,262144,26214
4,2097152,2097152,2097152,16777216,16777216,16777216,16777216,16777216,268435456,268435
456,1073741824,1073741824,1073741824,4294967296,8589934592,17179869184]}"
```

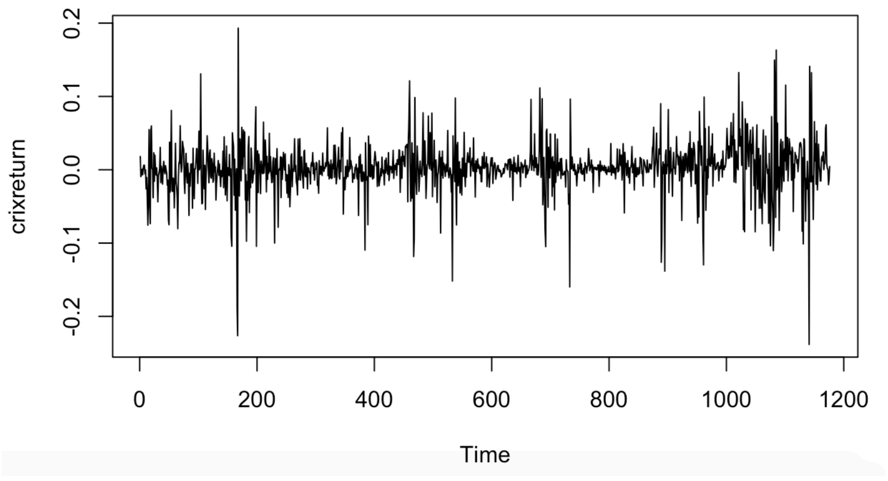
## 4. CRIX data

```
#install.packages("rjson", repos="http://cran.us.r-project.org")
library(rjson)
json_file = "http://crix.hu-berlin.de/data/crix.json"
json_data = fromJSON(file=json_file)
lst <- lapply(json_data,function(x){
   df<-data.frame(date=x$date,price=x$price)
   return(df)
})
```

```
crix_data_frame <- Reduce(rbind,lst)
plot(crix_data_frame$date,crix_data_frame$price)

# install.packages("forecast")
# install.packages("tseries")
library(forecast)
library(tseries)
ts.plot(crix_data_frame$price)
Acf(crix_data_frame$price)

for(i in 1:length(crix_data_frame$price)){
   crixreturn[i] <- log(crix_data_frame$price[i+1]/crix_data_frame$price[i])
}
ts.plot(crixreturn)
Box.test(crixreturn, type = "Ljung-Box", lag = 20)
autocorr = acf(crixreturn, lag.max = 20, ylab = "Sample Autocorrelation", main =
NA, lwd = 2, ylim = c(-0.3, 1))
Acf(crixreturn)
Pacf(crixreturn)
arima(crixreturn,order = c(2,0,2))
```

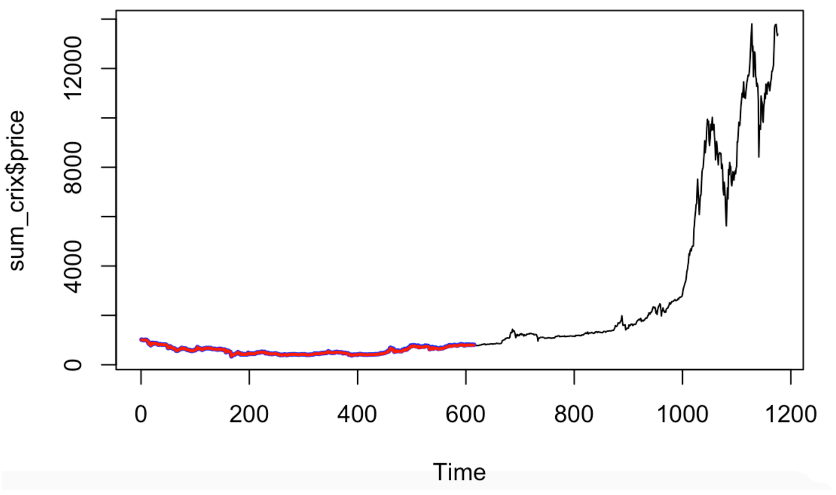


## Homework 4

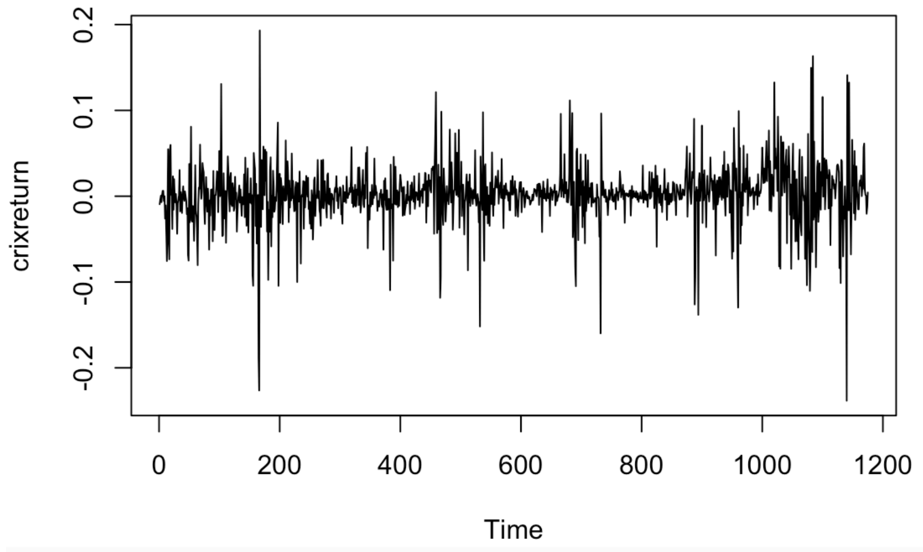
1. **Figure3,4,5,6**

```
#HW4.1
library(rjson)
json_file = "http://crix.hu-berlin.de/data/crix.json"
json_data = fromJSON(file=json_file)
```

```
lst <- lapply(json_data,function(x){
   df<-data.frame(date=x$date,price=x$price)
   return(df)
})
crix_data_frame <- Reduce(rbind,lst)
crix_data_frame <- crix_data_frame[-1,]
load(file = "ecrix.RData")
load(file = "efcrix.RData")
length(ecrix)=length(crix_data_frame$price)
length(efcrix)=length(crix_data_frame$price)
ecrix_data_frame <- as.data.frame(ecrix)
efcrix_data_frame <- as.data.frame(efcrix)
#install.packages("dplyr")
library(dplyr)
sum_crix <- cbind(crix_data_frame,ecrix_data_frame,efcrix_data_frame)
#figure3
ts.plot(sum_crix$price)
lines(sum_crix$price,col="black",lwd=0.5)
lines(sum_crix$ecrix,col="blue",lwd=1)
lines(sum_crix$efcrix,col="red",lwd=1)
```



```
#figure4
crixreturn <- diff(log(crix_data_frame$price))
ts.plot(crixreturn)
```



*#figure5*

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

*lines(density(crixreturn),lwd=1)*

*mu = mean(crixreturn)*

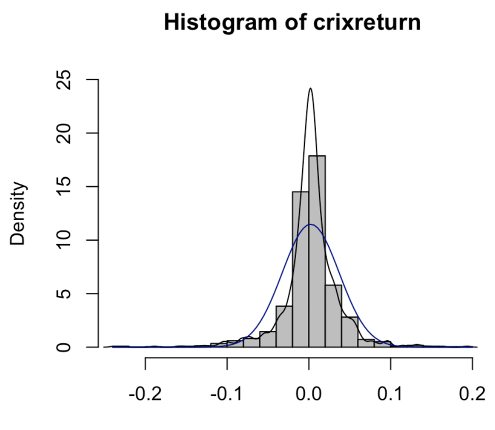
*sigma = sd(crixreturn)*

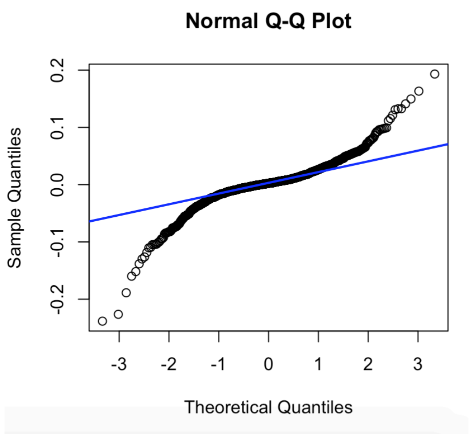
*x = seq(-4, 4, length = 100)*

*curve(dnorm(x, mean = mean(crixreturn), sd = sd(crixreturn)), add = TRUE, col = "darkblue", lwd = 1)*

*qqnorm(crixreturn)*

*qqline(crixreturn, col = "blue", lwd = 2)*



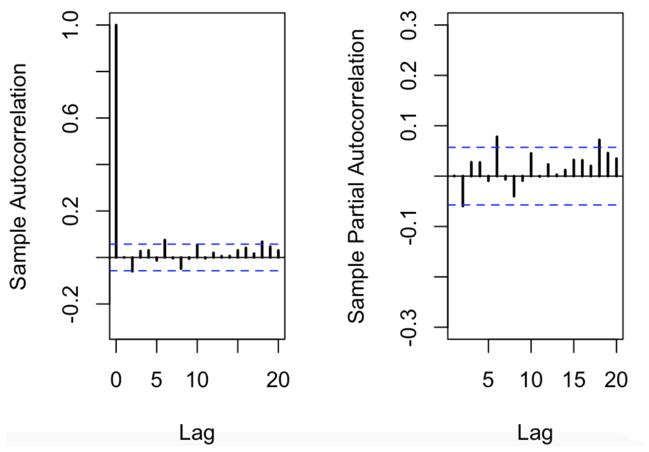


*#figure6*

*Box.test(crixreturn, type = "Ljung-Box", lag = 20)*

*adf.test(crixreturn, alternative = "stationary")*

```
kpss.test(crixreturn, null = "Trend")
par(mfrow = c(1, 2))
autocorr = acf(crixreturn, lag.max = 20, ylab = "Sample Autocorrelation", main
= NA, lwd = 2, ylim = c(-0.3, 1))
print(cbind(autocorr$lag, autocorr$acf))
Box.test(crixreturn, type = "Ljung-Box", lag = 1, fitdf = 0)
Box.test(autocorr$acf, type = "Ljung-Box")
autopcorr = pacf(crixreturn, lag.max = 20, ylab = "Sample Partial
Autocorrelation",main = NA, ylim = c(-0.3, 0.3), lwd = 2)
```

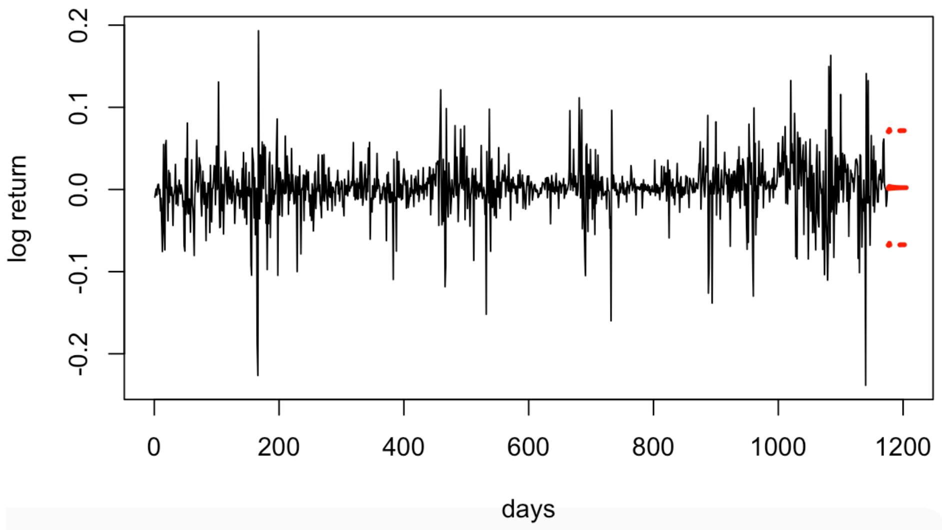


**2. Figure 7**

```
par(mfrow = c(1, 1))
auto.arima(crixreturn)
fit1 = arima(crixreturn, order = c(1, 0, 1))
tsdiag(fit1)
Box.test(fit1$residuals, lag = 1)

aic = matrix(NA, 6, 6)
for (p in 0:4) {
  for (q in 0:3) {
    a.p.q = arima(crixreturn, order = c(p, 0, q))
    aic.p.q = a.p.q$aic
    aic[p + 1, q + 1] = aic.p.q
  }
}
aic

bic = matrix(NA, 6, 6)
for (p in 0:4) {
    for (q in 0:3) {
        b.p.q = arima(crixreturn, order = c(p, 0, q))
        bic.p.q = AIC(b.p.q, k = log(length(crixreturn)))
        bic[p + 1, q + 1] = bic.p.q
    }
}
bic
fit4 = arima(crixreturn, order = c(2, 0, 3))
tsdiag(fit4)
Box.test(fit4$residuals, lag = 1)
fitr4 = arima(crixreturn, order = c(2, 1, 3))
tsdiag(fitr4)
Box.test(fitr4$residuals, lag = 1)
fit202 = arima(crixreturn, order = c(2, 0, 2))
tsdiag(fit202)
tsdiag(fit4)
tsdiag(fitr4)
AIC(fit202, k = log(length(crixreturn)))
AIC(fit4, k = log(length(crixreturn)))
AIC(fitr4, k = log(length(crixreturn)))
fit202$aic
fit4$aic
fitr4$aic
fit202 = arima(crixreturn, order = c(2, 0, 2))
crpre = predict(fit202, n.ahead = 30)
dates = seq(as.Date("02/08/2014", format = "%d/%m/%Y"), by = "days",
length = length(crixreturn))
plot(crixreturn, type = "l", xlim = c(0, 1200), ylab = "log return", xlab =
"days", lwd = 1)
lines(crpre$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)
```



log return
0.2
0.1
0.0
-0.1
-0.2
0
200
400
600
800
1000
1200
days