# Final Exam

*Reporter： Yang Liu*

*Instructor： Wolfgang Härdle*

# unit 1

# Logistic Regression

- In statistics, logistic regression, or logit regression, or logit model[1] is a regression model where the dependent variable (DV) is categorical. This article covers the case of a binary dependent variable—that is, where the output can take only two values, "0" and "1", which represent outcomes such as pass/fail, win/lose, alive/dead or healthy/sick. Cases where the dependent variable has more than two outcome categories may be analysed in multinomial logistic regression, or, if the multiple categories are ordered, in ordinal logistic regression.[2] In the terminology of economics, logistic regression is an example of a qualitative response/discrete choice model.

# Definition of the logistic function

An explanation of logistic regression can begin with an explanation of the standard logistic function. The logistic function is useful because it can take any real input t, ($t \epsilon \mathrm{R}$) , whereas the output always takes values between zero and one and hence is interpretable as a probability. The logistic function σ(t) is defined as follows:

$$\sigma(t) = \frac{e^t}{1+e^t} = \frac{1}{1+e^{-t}}$$

A graph of the logistic function on the t-interval (-6,6) is shown in Figure 1.

Let us assume that t is a linear function of a single explanatory variable x (the case where t is a linear combination of multiple explanatory variables is treated similarly). We can then express t as follows:

$$t = \beta_0 + \beta_1 x$$

And the logistic function can now be written as:

$$F(x) = \frac{1}{1+e^{-(\beta_0+\beta_1 x)}}$$

Note that F(x) is interpreted as the probability of the dependent variable equaling a "success" or "case" rather than a failure or non-case. It's clear that the response variables $Y_i$ are not identically distributed: $P(Y_i = 1 | X_i)$ differs from one data point $X_i$ to another, though they are independent given design matrix X and shared parameters β .

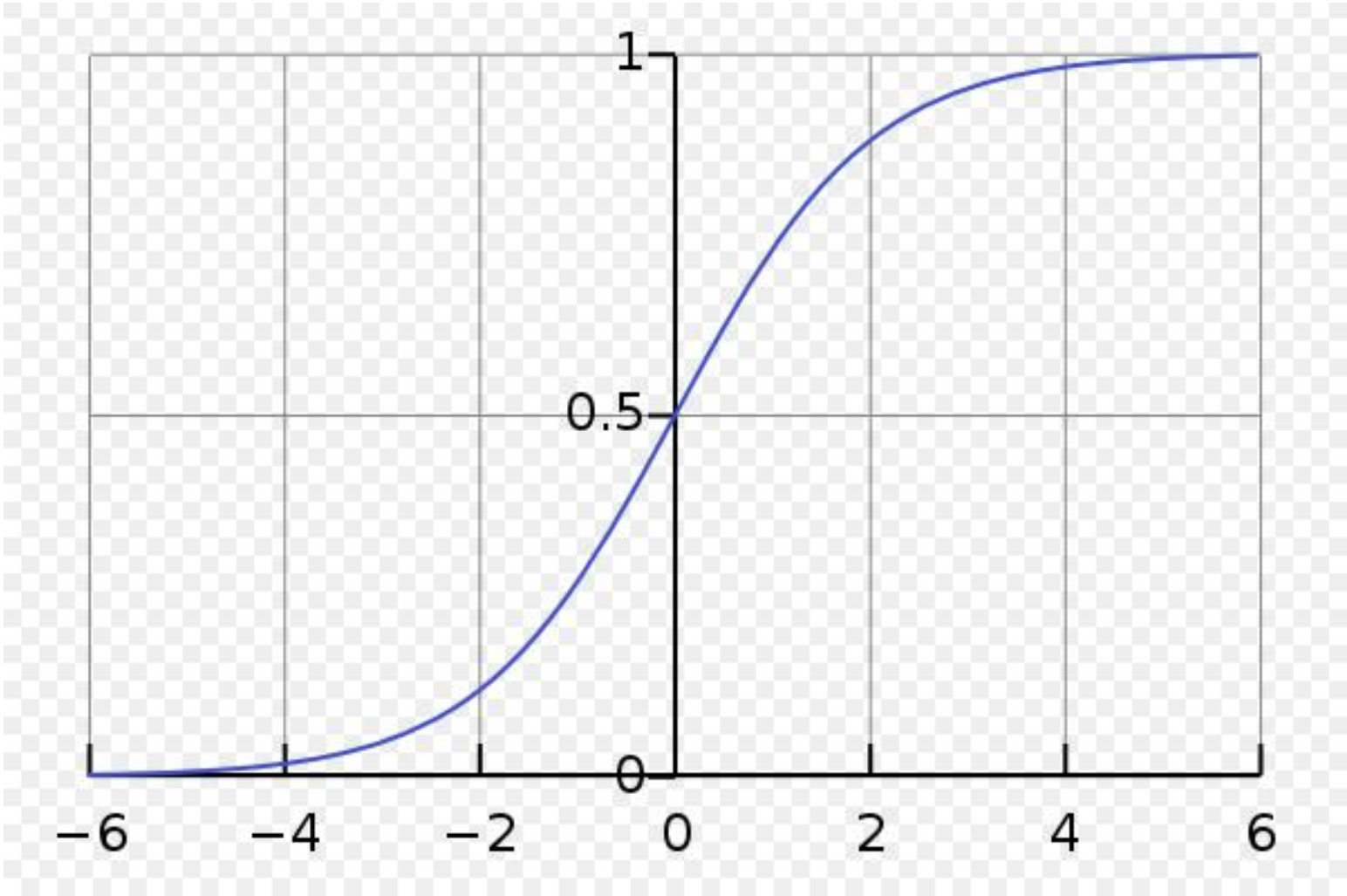



Figure 1

Logistic regression is used in various fields, including machine learning, most medical fields, and social sciences. For example, the Trauma and Injury Severity Score (TRISS), which is widely used to predict mortality in injured patients, was originally developed by Boyd et al. using logistic regression.[4] Many other medical scales used to assess severity of a patient have been developed using logistic regression.

C13

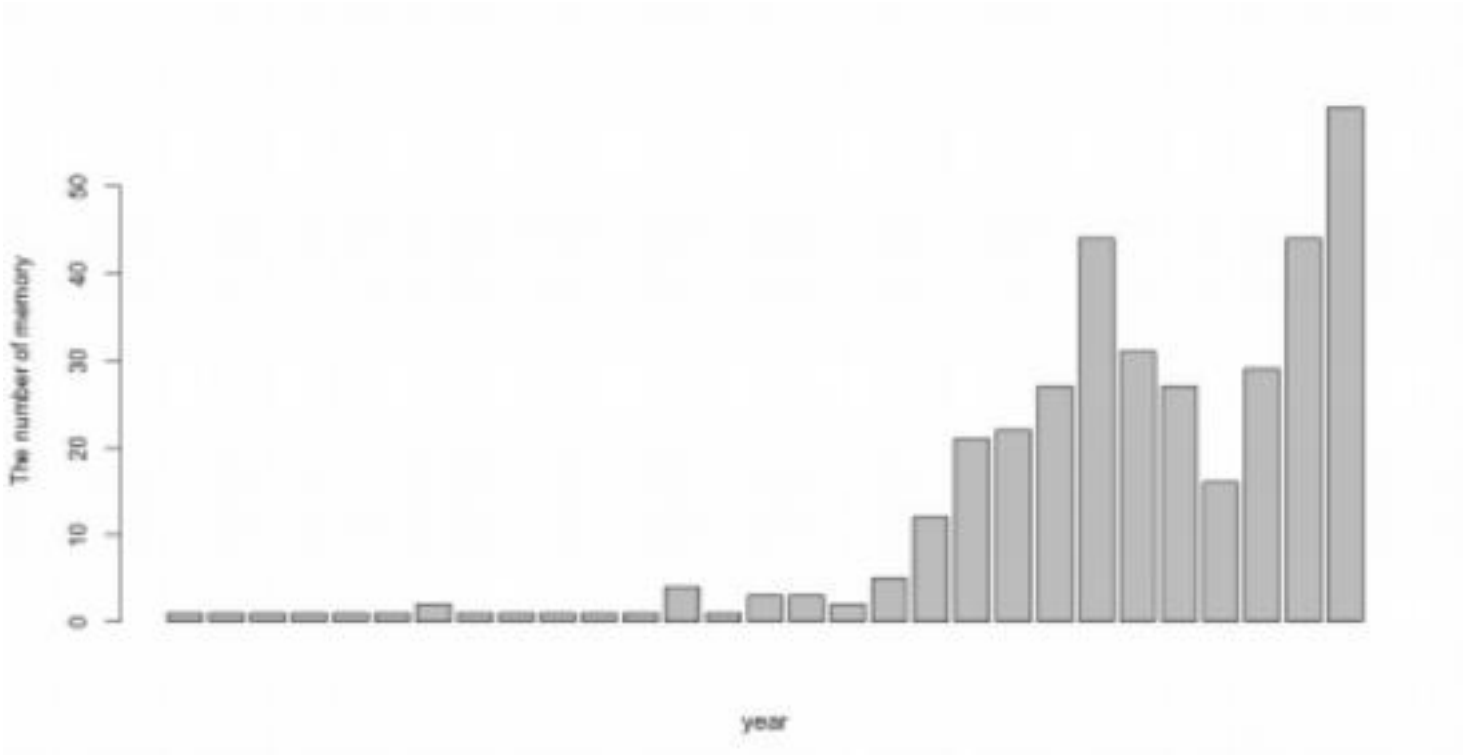
| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| 1 | Intro date | Processor | | intro date | count | | | |
| 2 | 1971 | 1971 4004 | | 1971 | 1 | | | |
| 3 | 1972 | 1972 8008 | | 1972 | 1 | | | |
| 4 | 1974 | 1974 8080 | | 1974 | 1 | | | |
| 5 | 1976 | 1976 8085 | | 1976 | 1 | | | |
| 6 | 1978 | 1978 8086 | | 1978 | 1 | | | |
| 7 | 1979 | 1979 8088 | | 1979 | 1 | | | |
| 8 | 1982 | 1982 80286 | | 1982 | 2 | | | |
| 9 | 1982 | 1982 80186 | | 1985 | 1 | | | |
| 10 | 1985 | 1985 Intel386™ DX Processor | | 1988 | 1 | | | |
| 11 | 1988 | 1988 Intel386™ SX Processor | | 1989 | 1 | | | |
| 12 | 1989 | 1989 Intel486™ DX Processor | | 1990 | 1 | | | |
| 13 | 1990 | 1990 Intel386™ SL Processor | | 1991 | 1 | | | |
| 14 | 1991 | 1991 Intel486™ SX Processor | | 1992 | 4 | | | |
| 15 | 1992 | 1992 Intel486™ SL Processor | | 1993 | 1 | | | |
| 16 | 1992 | 1992 Intel486™ SX Processor | | 1994 | 3 | | | |
| 17 | 1992 | 1992 IntelDX2™ Processor | | 1995 | 3 | | | |
| 18 | 1992 | 1992 IntelDX2™ Processor | | 1996 | 2 | | | |
| 19 | 1993 | 1993 Intel® Pentium® Processor | | 1997 | 5 | | | |
| 20 | 1994 | 1994 Intel® Pentium® Processor | | 1998 | 12 | | | |
| 21 | 1994 | 1994 IntelDX4™ Processor | | 1999 | 21 | | | |
| 22 | 1994 | 1994 Intel® Pentium® Processor | | 2000 | 22 | | | |
| 23 | 1995 | 1995 Intel® Pentium® Pro Processor | | 2001 | 27 | | | |
| 24 | 1995 | 1995 Intel® Pentium® Processor | | 2002 | 44 | | | |
| 25 | 1995 | 1995 Intel® Pentium® Processor | | 2003 | 31 | | | |
| 26 | 1996 | 1996 Intel® Pentium® Processor | | 2004 | 27 | | | |

# unit 2

# Code

```
year<-c(1971,1972,1974,1976,1978,1979,1982,1985,1988,1989,1990,1991,1992,1993,
        1994,1995,1996,1997,1998,1999,2000,2001,2002,2003,2004,2005,2006,2007,2008)
number<-c(1,1,1,1,1,1,2,1,1,1,1,1,4,1,3,3,2,5,12,21,22,27,44,31,27,16,29,44,59)
#HW1#
plot(year,number,type = "b",
     col="black",main = "The history of computer memory",
     sub = "This is the change in the number of types of computer memory",
     xlab = "year",ylab = "The number of memory")
barplot(number,
        xlab = "year",ylab ="The number of memory ")

#HW2#
#HW3#
lambda=2
x=seq(0:6)
P<-data.frame(dpois(x,lambda))
sum<-(P[1,]+P[7,]+P[2,]+P[6,]+P[3,]+P[5,]+P[4,]+P[4,])
sum
lambda=5
x=0
dpois(x,lambda)
```

# Result




```
> x = 6
> n = 1000
> lambda = 2
> p = lambda / n
> dbinom (x,2*n,p) # binomial probability mass function
[1] 0.1042477
> dpois (x, 2*lambda ) # Poisson probability mass function
[1] 0.1041956
> dpois (0, 5 )
[1] 0.006737947
```

# unit 3

# Code

#HW3-1#

- install.packages("digest", repos='http://cran.us.r-project.org')
- library("digest")
- s1=digest("I learn a lot from this class when I am proper listening to the professor","sha256")
- s2=digest("I do not learn a lot from this class when I am absent and playing on my Iphone","sha256")

#HW3-3#

- library(RJSONIO)
- letter<-LETTERS[1:9]
- color<-c("black","white","red","orange",  "yellow","blue","","green","navy","violet")
- data<-data.frame(letter,color)
- da<-as.matrix(data)
- cat(toJSON(da))

# Code

#HW3-4#

- 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)
- crix_data_frame=as.data.frame(json_data)
- crix_data_frame_t<-t(crix_data_frame)
- time<-crix_data_frame_t[seq(1,2350,by=2)]
- price<-crix_data_frame_t[seq(2,2350,by=2)]
- crix_data_frame<-cbind(time,price)
- time_series<-ts(data=price,start =c(2014,7,31),frequency = 365)
- plot(time_series)
- install.packages("tseries")
- library(tseries)
- adf.test(time_series)
- #Because p-value is greater than printed p-value, we can't reject the hypothesis#

# Result

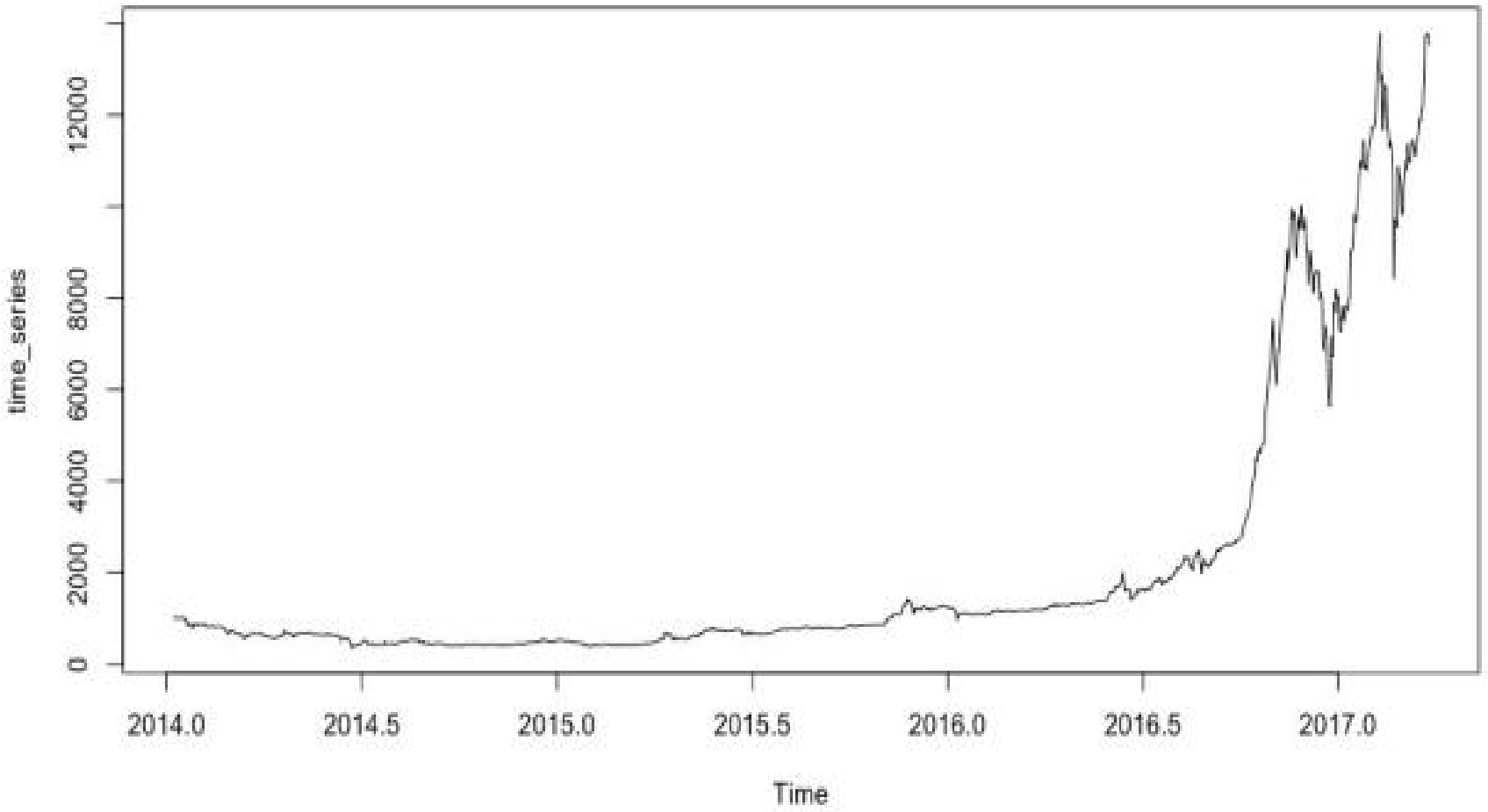
**HW3-1**

```
> s1
[1] "c16700de5a5c1961e279135f2be7dcf9c187cb6b21ac8032308c715e1ce9964c"
> s2
[1] "2533d529768409d1c09d50451d9125fdbaa6e5fd4efdeb45c04e3c68bcb3a63e"
```

**HW3-3**

```
[ {"letter": "A","color": "black" },{"letter": "B","color": "white" },
{"letter": "C","color": "red" },{"letter": "D","color": "orange" },
{"letter": "E","color": "yellow" },{"letter": "F","color": "blue" },
{"letter": "G","color": "green" },{ "letter": "H","color": "navy" },
{ "letter": "I","color": "violet" }]
```

# HW3-4


time_series
0
2000
4000
6000
8000
12000
2014.0
2014.5
2015.0
2015.5
2016.0
2016.5
2017.0
Time

# DSA(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).[1]

- Digital signatures are a standard element of most cryptographic protocol suites, and are commonly used for software distribution, financial transactions, contract management software, and in other cases where it is important to detect forgery or tampering.

# Applications of digital signatures

- Authentication

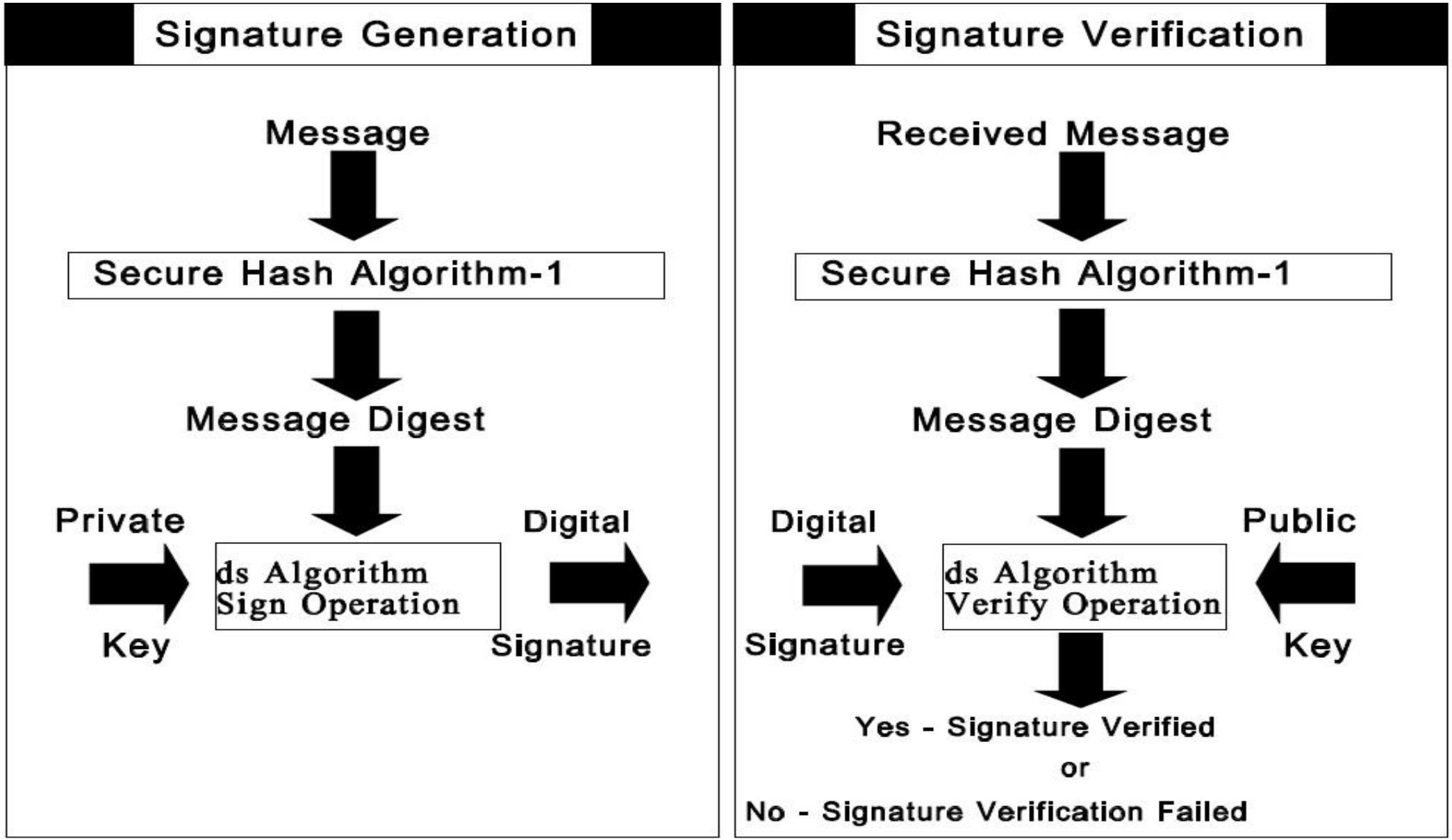
Digital signatures can be used to authenticate the source of messages. When ownership of a digital signature secret key is bound to a specific user, a valid signature shows that the message was sent by that user.

- Integrity

if a message is digitally signed, any change in the message after signature invalidates the signature.

- Non-repudiation

An entity that has signed some information cannot at a later time deny having signed it. Similarly, access to the public key only does not enable a fraudulent party to fake a valid signature.


Signature Generation
Message
Secure Hash Algorithm-1
Message Digest
Private
Digital
ds Algorithm
Sign Operation
Key
Signature
Signature Verification
Received Message
Secure Hash Algorithm-1
Message Digest
Digital
Public
ds Algorithm
Verify Operation
Signature
Key
Yes - Signature Verified
or
No - Signature Verification Failed

# Signing and Verifying

Let $H$ be the hashing function and $m$ the message:

- Generate a random per-message value $k$ where $1 < k < q$
- Calculate $r = \left(g^k \bmod p\right) \bmod q$
- In the unlikely case that $r = 0$, start again with a different random $k$
- Calculate $s = k^{-1}\left(H(m) + xr\right) \bmod q$
- In the unlikely case that $s = 0$, start again with a different random $k$
- The signature is $(r, s)$

- Reject the signature if $0 < r < q$ or $0 < s < q$ is not satisfied.
- Calculate $w = s^{-1} \bmod q$
- Calculate $u_1 = H(m) \cdot w \bmod q$
- Calculate $u_2 = r \cdot w \bmod q$
- Calculate $v = \left(g^{u_1} y^{u_2} \bmod p\right) \bmod q$
- The signature is invalid unless $v = r$

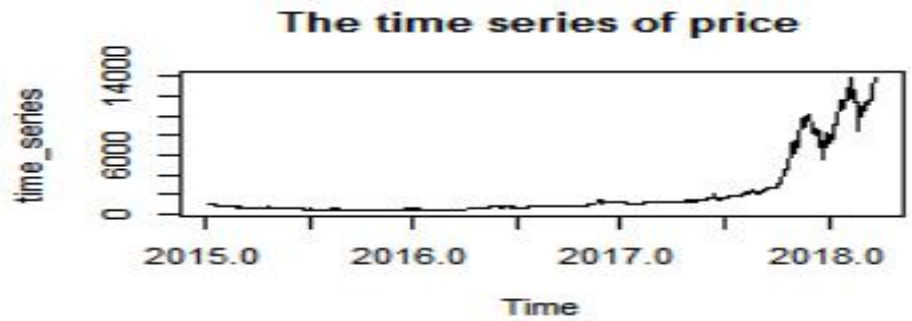
# unit 4

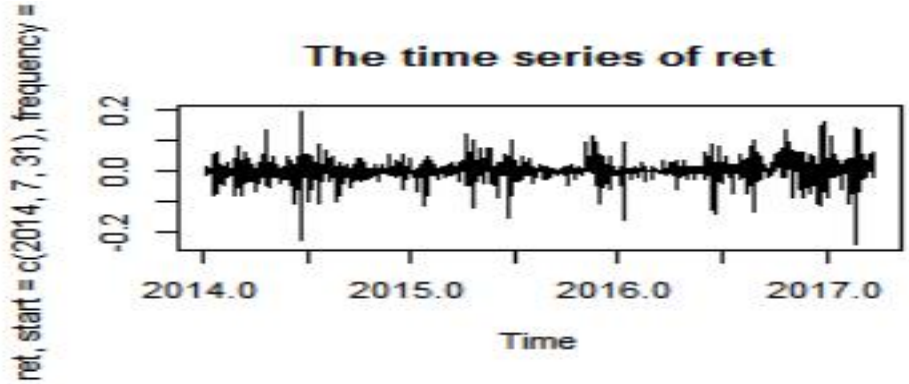
# HW 4-1

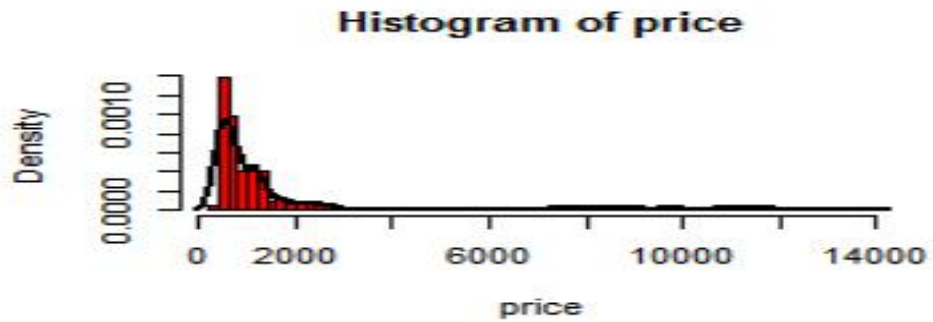
## *code*

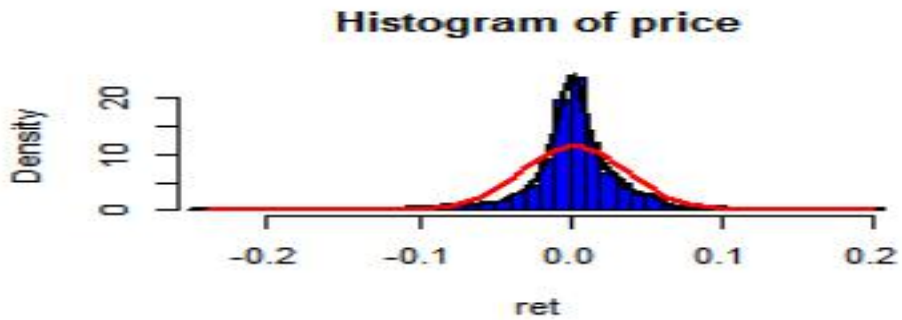
```
install.packages("rjson",repos = "http://cran.us.r-project.org")
library("rjson")
json_file="http://crix.hu-berlin.de/data/crix.json"
json_data=fromJSON(file=json_file)
crix_data_frame=as.data.frame(json_data)
#plot of time series#
par(mfrow=c(3,2))
crix_data_frame_t<-t(crix_data_frame)
time<-crix_data_frame_t[seq(1,2350,by=2)]
price<-as.numeric(crix_data_frame_t[seq(2,2350,by=2)])
crix_data_frame<-cbind(time,price)
time_series<-ts(data=price,start =c(2015,7,31),frequency = 365)
plot(time_series,main="The time series of price")
#plot of return#
ret<-diff(log(price))
plot(ts(ret,start= c(2015,7,31),frequency = 365),main="The time series of ret")
#histogram of price and ret#
hist(price,breaks=50,col = "red",freq = FALSE,xlab="price",main = "Histogram of
price")
lines(density(price),col="black",lwd=2)

hist(ret,breaks = 40,col = "blue",freq = FALSE,main = "Histogram of price")
lines(density(ret),col="black",lwd=2)

x=seq(-4,4,length=100)
curve(dnorm(x,mean=mean(ret),sd=sd(ret)),add=TRUE,col="red",lwd=2)
#normal qq plot#
qqnorm(ret)
qqline(ret,col="blue",lwd=3)
```

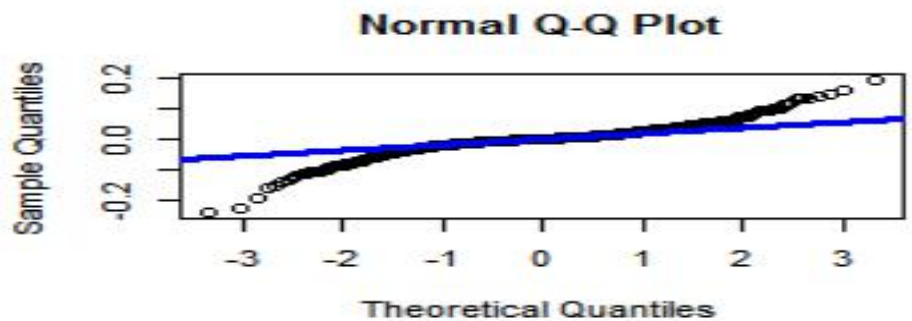
# *Result*


The time series of price
time_series
0
6000
14000
2015.0
2016.0
2017.0
2018.0
Time


The time series of ret
ret, start = c(2014, 7, 31), frequency =
-0.2
0.0
0.2
2014.0
2015.0
2016.0
2017.0
Time


Histogram of price
Density
0.0000
0.0010
0
2000
6000
10000
14000
price


Histogram of price
Density
0
10
20
-0.2
-0.1
0.0
0.1
0.2
ret

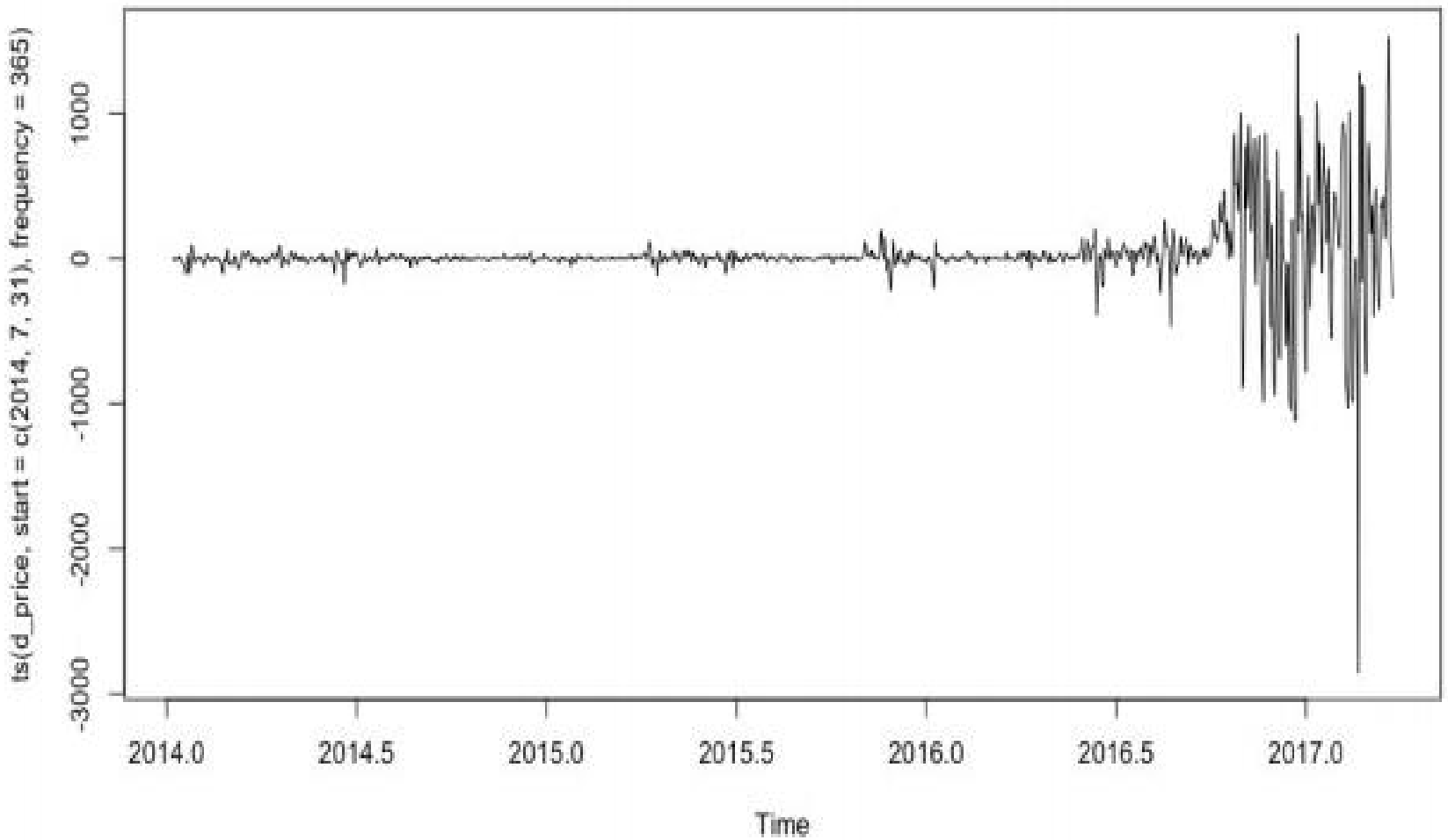

Normal Q-Q Plot
Sample Quantiles
-0.2
0.0
0.2
-3
-2
-1
0
1
2
3
Theoretical Quantiles

# HW 4-2 andHW 4-3

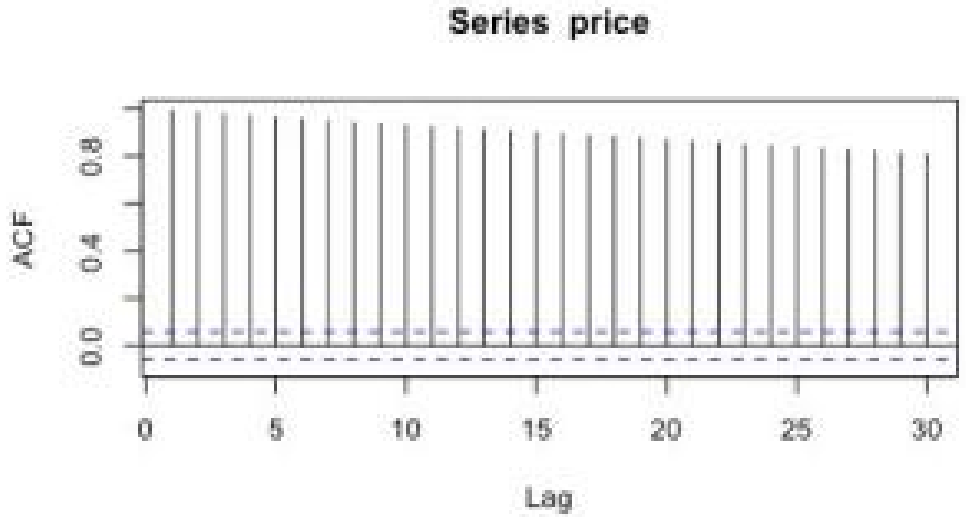
*code*

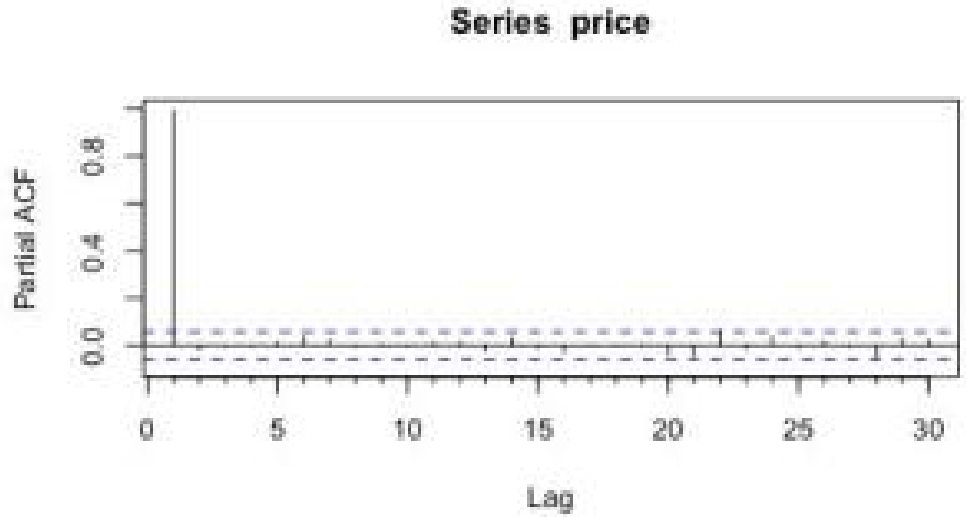
```
library(forecast)
library(tseries)
#stationary test#
ndiffs(price)
par(mfrow=c(1,1))
d_price<-diff(price,lag=2)
plot(ts(d_price,start =c(2015,7,31),frequency = 365))
adf.test(d_price)
#ACF and PACF#
par(mfrow=c(2,2))
Acf(price)
Pacf(price)
Acf(d_price)
Pacf(d_price)
#fit model#
fit<-arima(time_series,order = c(0,2,3))
fit
accuracy(fit)
#model test#
par(mfrow=c(1,1))
qqnorm(fit$residuals)
qqline(fit$residuals)
Box.test(fit$residuals,type = "Ljung-Box")
#model forecast#
forecast(fit,5)
plot(forecast(fit,5),xlab = "time",ylab = "price")
```

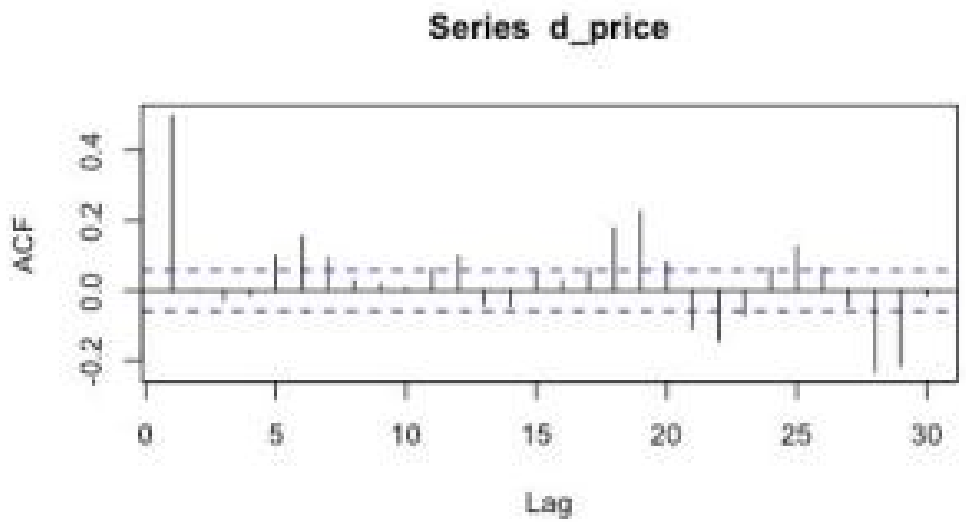
# *Result*

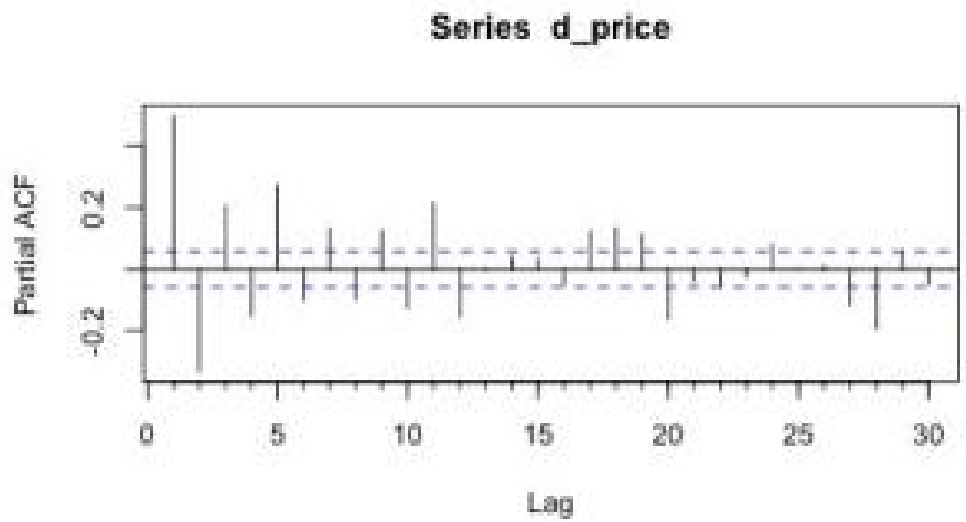

ts(d_price, start = c(2014, 7, 31), frequency = 365)
1000
0
-1000
-2000
-3000
2014.0
2014.5
2015.0
2015.5
2016.0
2016.5
2017.0
Time

# *Result*

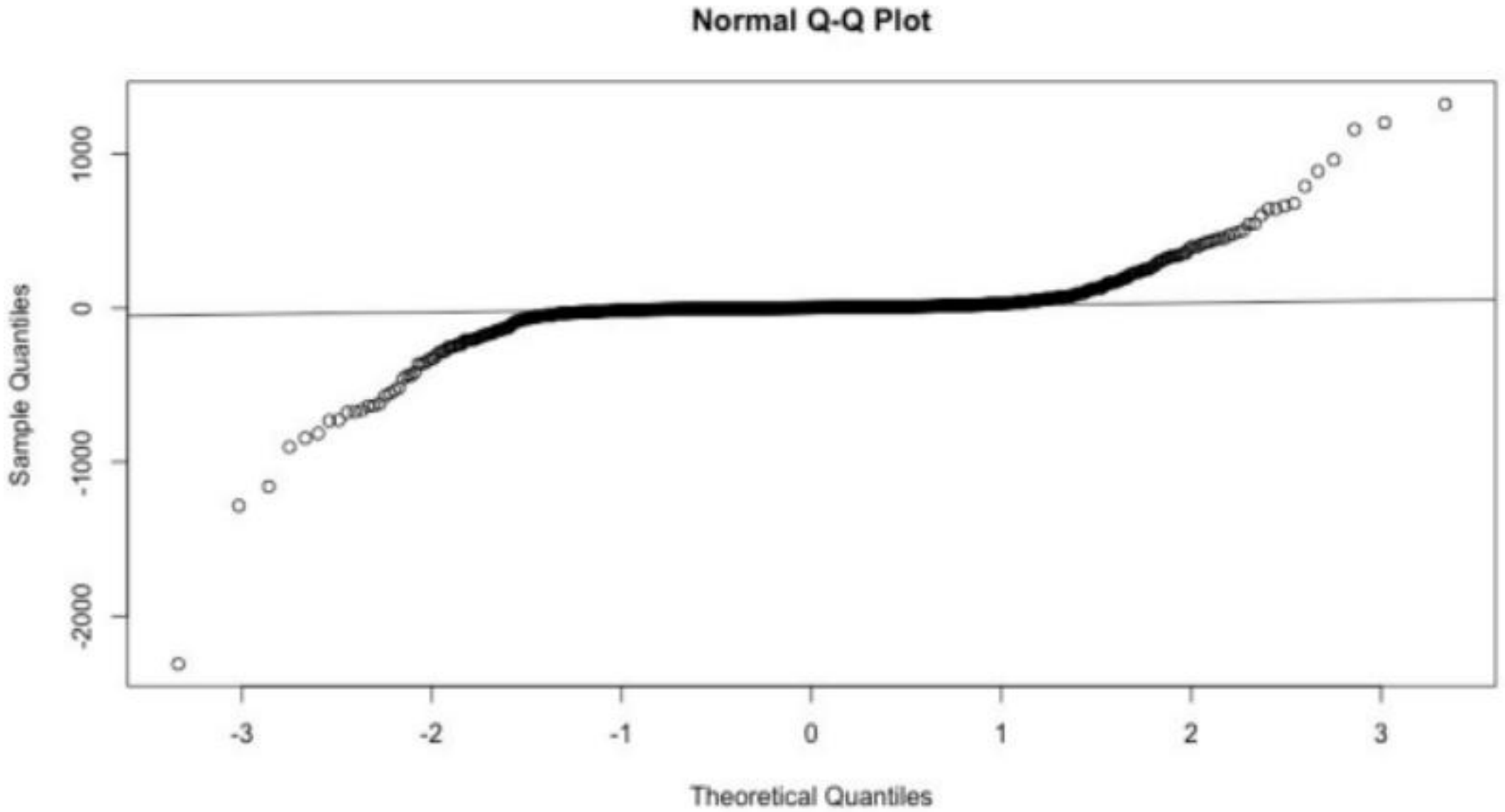

Series price
ACF
0.0
0.4
0.8
0
5
10
15
20
25
30
Lag


Series price
Partial ACF
0.0
0.4
0.8
0
5
10
15
20
25
30
Lag

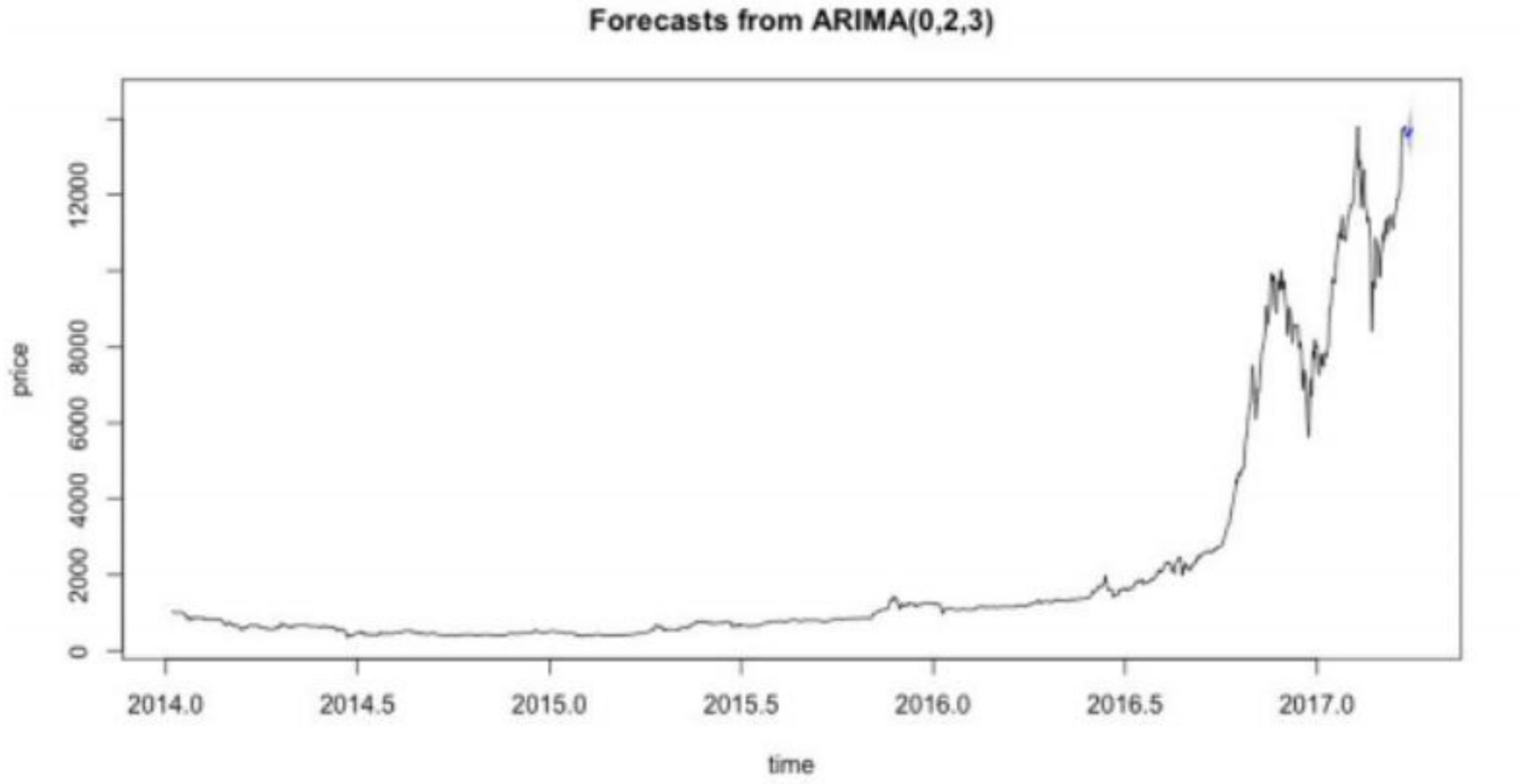

Series d_price
ACF
-0.2
0.0
0.2
0.4
0
5
10
15
20
25
30
Lag


Series d_price
Partial ACF
-0.2
0.2
0
5
10
15
20
25
30
Lag

# *Result*


Normal Q-Q Plot
Sample Quantiles
1000
0
-1000
-2000
-3
-2
-1
0
1
2
3
Theoretical Quantiles

# *Result*


Forecasts from ARIMA(0,2,3)
price
0
2000
4000
6000
8000
12000
2014.0
2014.5
2015.0
2015.5
2016.0
2016.5
2017.0
time

# *Result*

```
Call:
arima(x = time_series, order = c(0, 2, 3))

Coefficients:
          ma1     ma2      ma3
      -1.0186  0.0286  -0.0033
s.e.   0.0293  0.0417   0.0308

sigma^2 estimated as 29389:  log likelihood = -7700.73,  aic = 15409.45
> accuracy(fit)
                    ME     RMSE      MAE       MPE     MAPE      MASE         ACF1
Training set 5.963777 171.2866 61.05574 0.1945169 2.177519 0.9829255 -0.002283453
```

| | Point Forecast | Lo 80 | Hi 80 | Lo 95 | Hi 95 |
|---|---|---|---|---|---|
| 2017.2356 | 13559.25 | 13339.55 | 13778.95 | 13223.25 | 13895.25 |
| 2017.2384 | 13601.01 | 13293.19 | 13908.83 | 13130.24 | 14071.78 |
| 2017.2411 | 13643.86 | 13266.78 | 14020.94 | 13067.16 | 14220.55 |
| 2017.2438 | 13686.71 | 13250.52 | 14122.90 | 13019.62 | 14353.80 |
| 2017.2466 | 13729.56 | 13240.71 | 14218.40 | 12981.93 | 14477.18 |