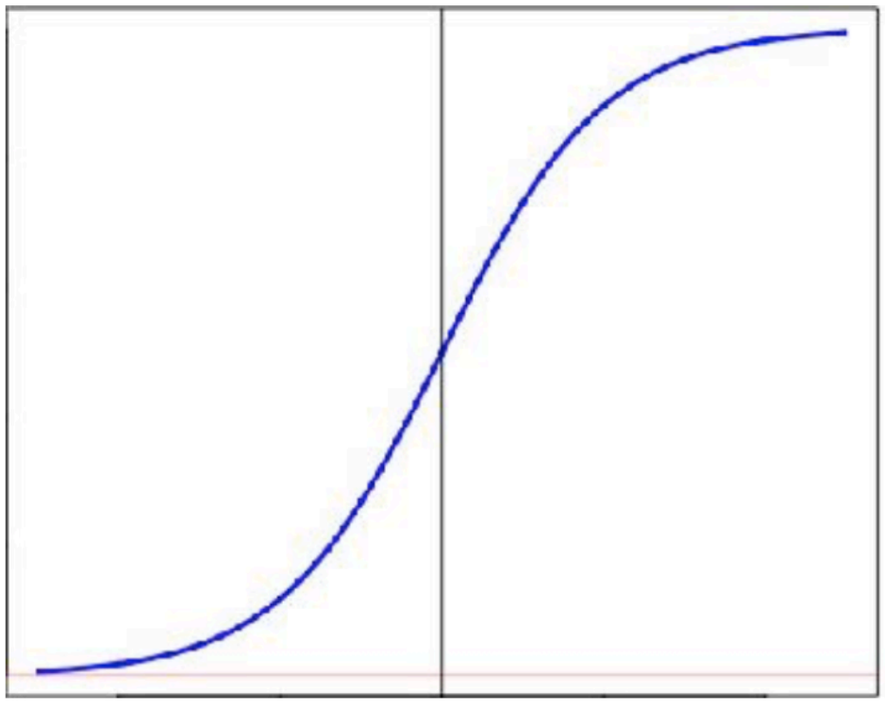
Final EXAM

Hanjun Hua
27720161153020

Uuit 1



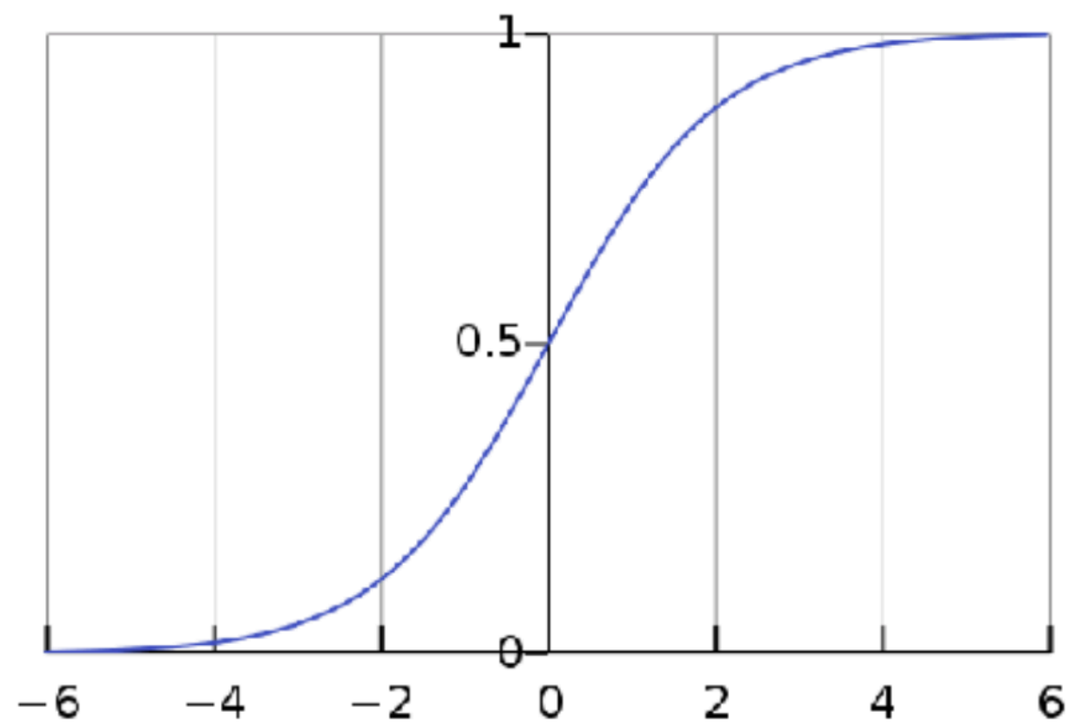
Logistic regression model is a classification model. There are many similarities between the logistic regression model and multiple linear regression, the biggest difference is the dependent variable. Both of these are generalized linear models.

Definition:Logistic regression model is the following conditional probability distribution：

$$P(Y=1 \mid x)=\frac{\exp(-g(x))}{1+\exp(-g(x))}$$

$$P(Y=0 \mid x)=\frac{1}{1+\exp(-g(x))}$$

Logistic regression measures the relationship between the categorical dependent variable and one or more independent variables by estimating probabilities using a logistic function, which is the cumulative logistic distribution. Thus, it treats the same set of problems as probit regression using similar techniques, with the latter using a cumulative normal distribution curve instead.



Logistic regression is mainly applied in epidemiology. Such as to explore the risk factors of certain diseases, and predict the probability of a disease according to the risk factors. In addition, it can be used survival analysis, hydrology and economics.

粘贴
等线 Regular (正文)
12
常规
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			
27	1996	1996 Intel® Pentium® Processor		2005	16			
28	1997	1997 Mobile Intel® Pentium® Processor		2006	29			
29	1997	1997 Intel® Pentium® Pro Processor		2007	44			
30	1997	1997 Intel® Pentium® Processor		2008	59			

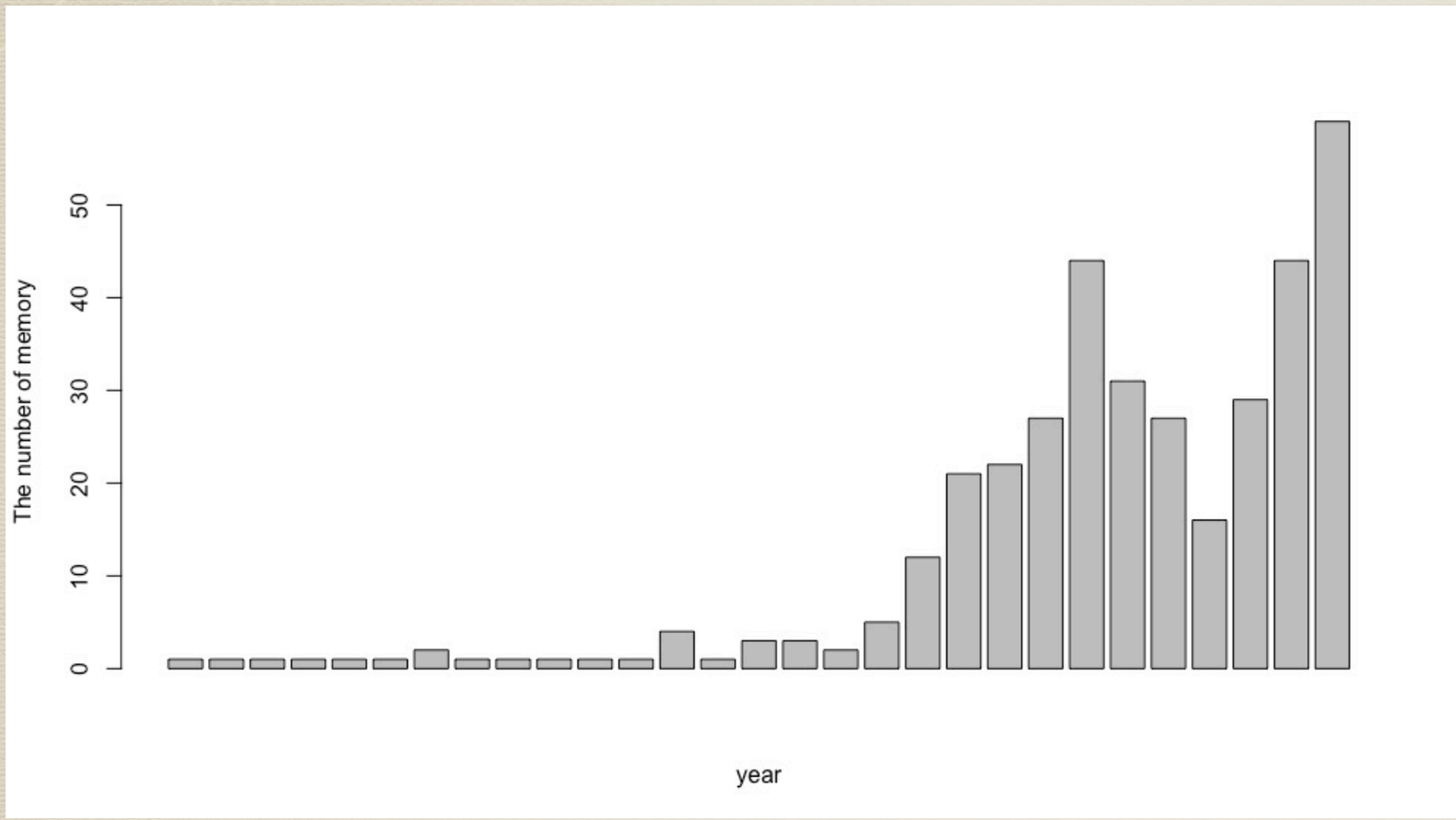
Uuit 2

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 ")

#HW3#
lambda=4
x=6
dpois(x,lambda)
lambda=5
x=0
dpois(x,lambda)
```

Result

HW2-1



HW2-3

```
> x=6
> dpois(x,lambda)
[1] 0.1041956
> lambda=5
> x=0
> dpois(x,lambda)
[1] 0.006737947
>
```

Uuit 3

Code

```
#HW3_1#
install.packages("digest",repos ='http://cran.us.r-project.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")

#HW3-3#
library("RJSONIO")
number<-c(1010,1011,1012,1013,1014,1015,1016)
name<-c("Alice","Bob","Cindy","David","Ella","Frank","Gres")
da<-as.matrix(data.frame(number,name))
s<-cat(toJSON(da))
writeLines(da,"name.json")

#HW3-4#
#download 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)
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)
#ARMA or ARIMA?#
library(tseries)
adf.test(time_series)
#Since p-value greater than printed p-value,we can't reject the hypothesis#
#the time series(time_series) is not stationary#
```

Result

HW3-1

```
> library("digest")
> digest("I learn a lot from this class when I am proper listening to the professor","sha256")
[1] "c16700de5a5c1961e279135f2be7dcf9c187cb6b21ac8032308c715e1ce9964c"
> digest("I do not learn a lot from this class when I am absent and playing on my Iphone","sha256")
[1] "2533d529768409d1c09d50451d9125fdbaa6e5fd4efdeb45c04e3c68bcb3a63e"
>
```

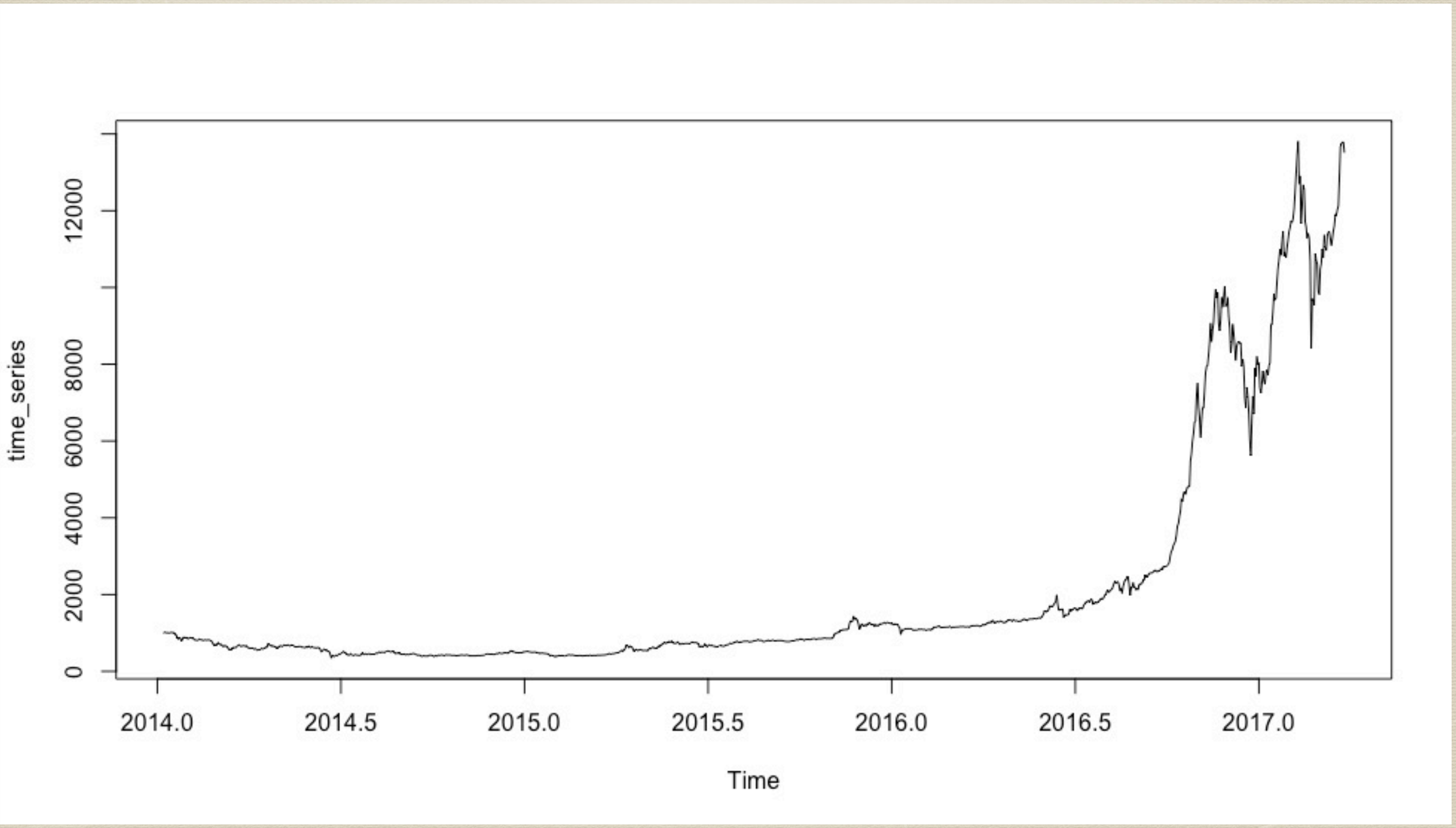
HW3-3

name.json

```
1010
1011
1012
1013
1014
1015
1016
Alice
Bob
Cindy
David
Ella
Frank
Gres
```

Result

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

JSON DATA

1. Create a JSON data set

```
library(rjsonio)
number<-c(1010,1011,1012,1013,1014,1015,1016)
name<-c("Alice","Bob","Cindy","David","Ella","Frank","Gres")
da<-as.matrix(data.frame(number,name))
s<-cat(toJSON(da))
```

Note: JSON data is a key-value pairs list.

2. Read the JSON data set

```
library("rjson")
json_data = fromJSON(file=data)
```

We can use function ‘fromJSON’ to read the JSON data set

Digital Signature Algorithm (DSA)

Hanjun Hua
27720161153020

Introduction of 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.

Key generation

Key generation included parameter generation and per-user keys.

Key generation has two phases. The first phase is a choice of algorithm parameters which may be shared between different users of the system, while the second phase computes public and private keys for a single user.

Signing

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

Verifying

- 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 $v=(g^{u1}y^{u2} \bmod p) \bmod q$
- The signature is invalid unless $v=r$

DSA is similar to the ElGamal signature scheme.

Sensitvity

With DSA, the entropy, secrecy, and uniqueness of the random signature value k are critical. It is so critical that violating any one of those three requirements can reveal the entire private key to an attacker. Using the same value twice (even while keeping k secret), using a predictable value, or leaking even a few bits of k in each of several signatures, is enough to reveal the private key x.

Uuit 4

Code

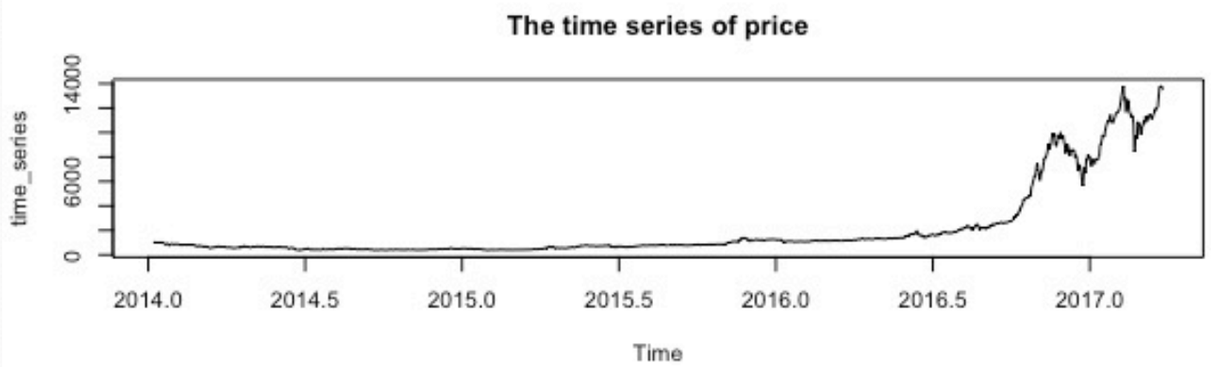
```
#HW4-1#
#download data#
install.packages("rjson",repos = "http://cran.us.r-project.org")
library("rjson")
json_file="http://crix.hu-berlin.de/data/crix.json"
json_data=fromJSON(file=json_file)
crix_data_frame=as.data.frame(json_data)
#polt of time series #
par(mfrow=c(3,2))
crix_data_frame_t<-t(crix_data_frame)
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(2014,7,31),frequency = 365)
plot(time_series,main="The time series of price")
#plot of  return  #
ret<-diff(log(price))
plot(ts(ret,start= c(2014,7,31),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)

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")
```

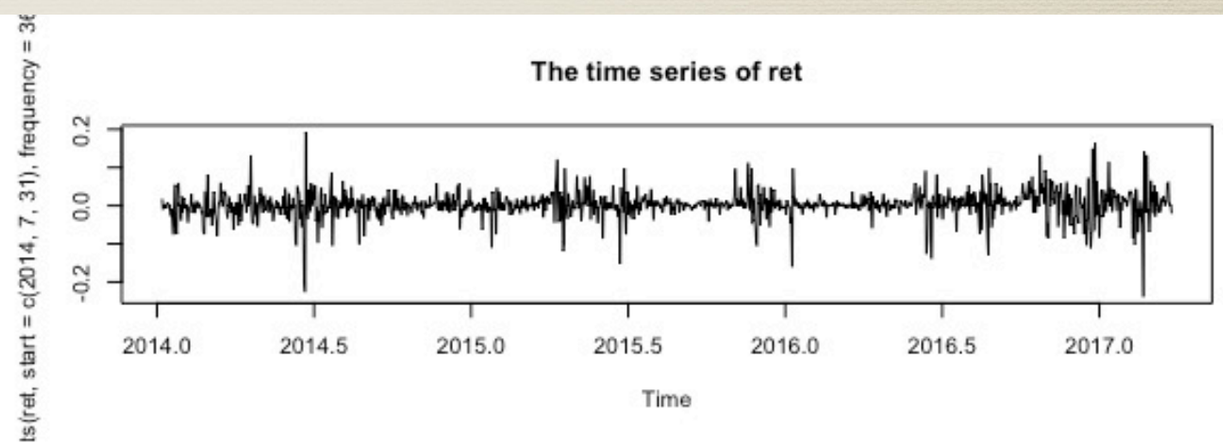
Code

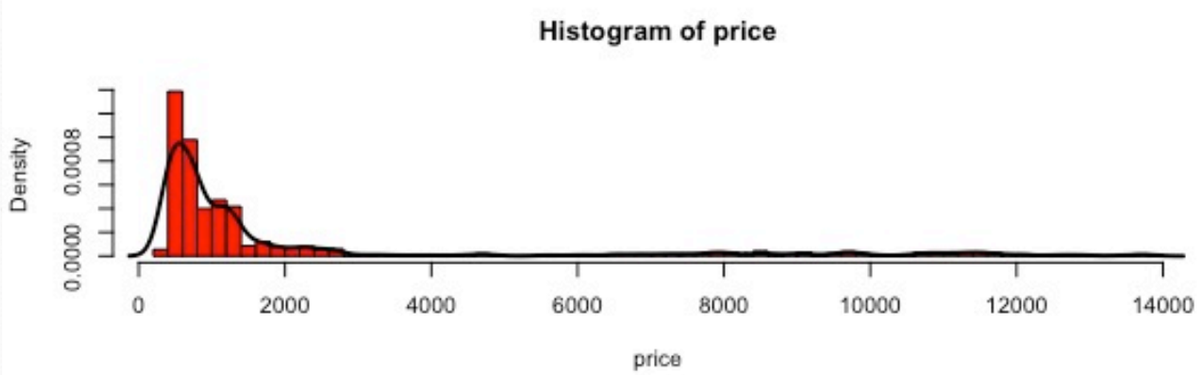
```
#  HW4-2 and HW4_3  #
library(forecast)
library(tseries)
#stationary test#
ndiffs(price)
par(mfrow=c(1,1))
d_price<-diff(price,lag=2)
plot(ts(d_price,start =c(2014,7,31),frequency = 365))
adf.test(d_price)
#ACF and PACF#
par(mfrow=c(2,2))
Acf(price)
Pacf(price)
Acf(d_price)
Pacf(d_price)
#fit model#
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")
```

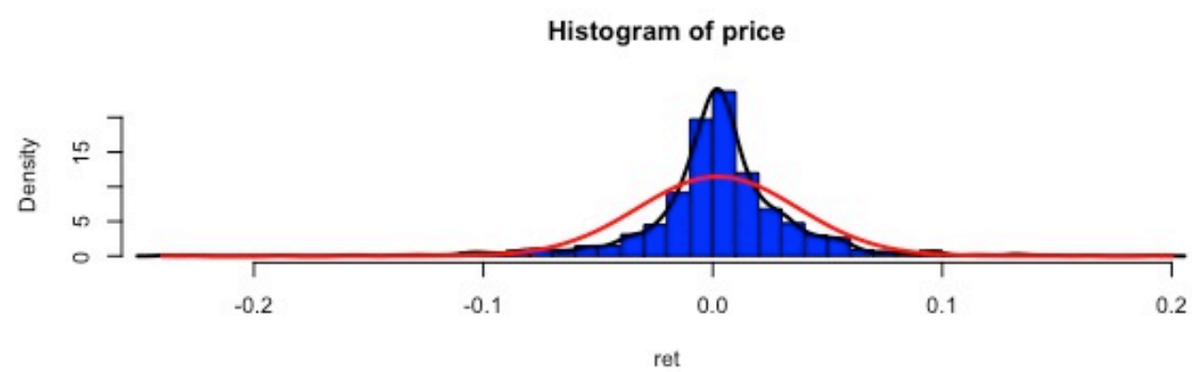
Result

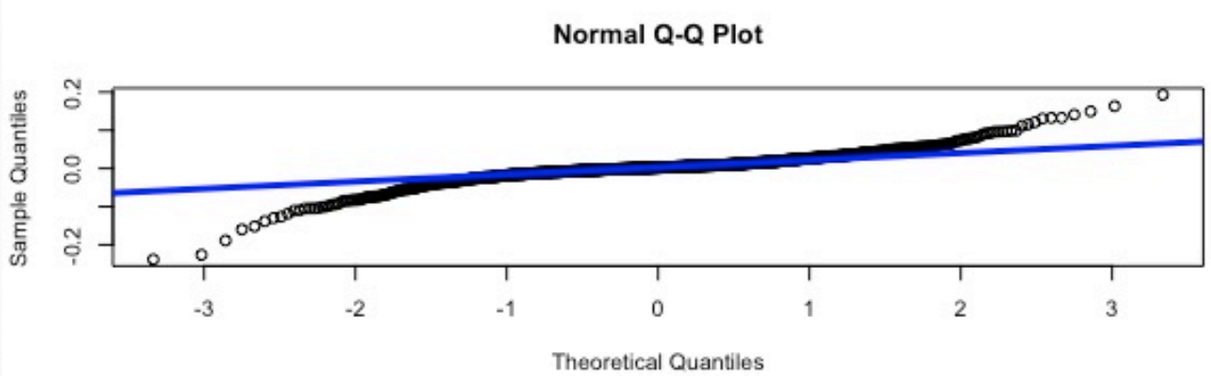
HW4-1


The time series of price
time_series
0
6000
14000
2014.0
2014.5
2015.0
2015.5
2016.0
2016.5
2017.0
Time


The time series of ret
ts(ret, start = c(2014, 7, 31), frequency = 36
-0.2
0.0
0.2
2014.0
2014.5
2015.0
2015.5
2016.0
2016.5
2017.0
Time

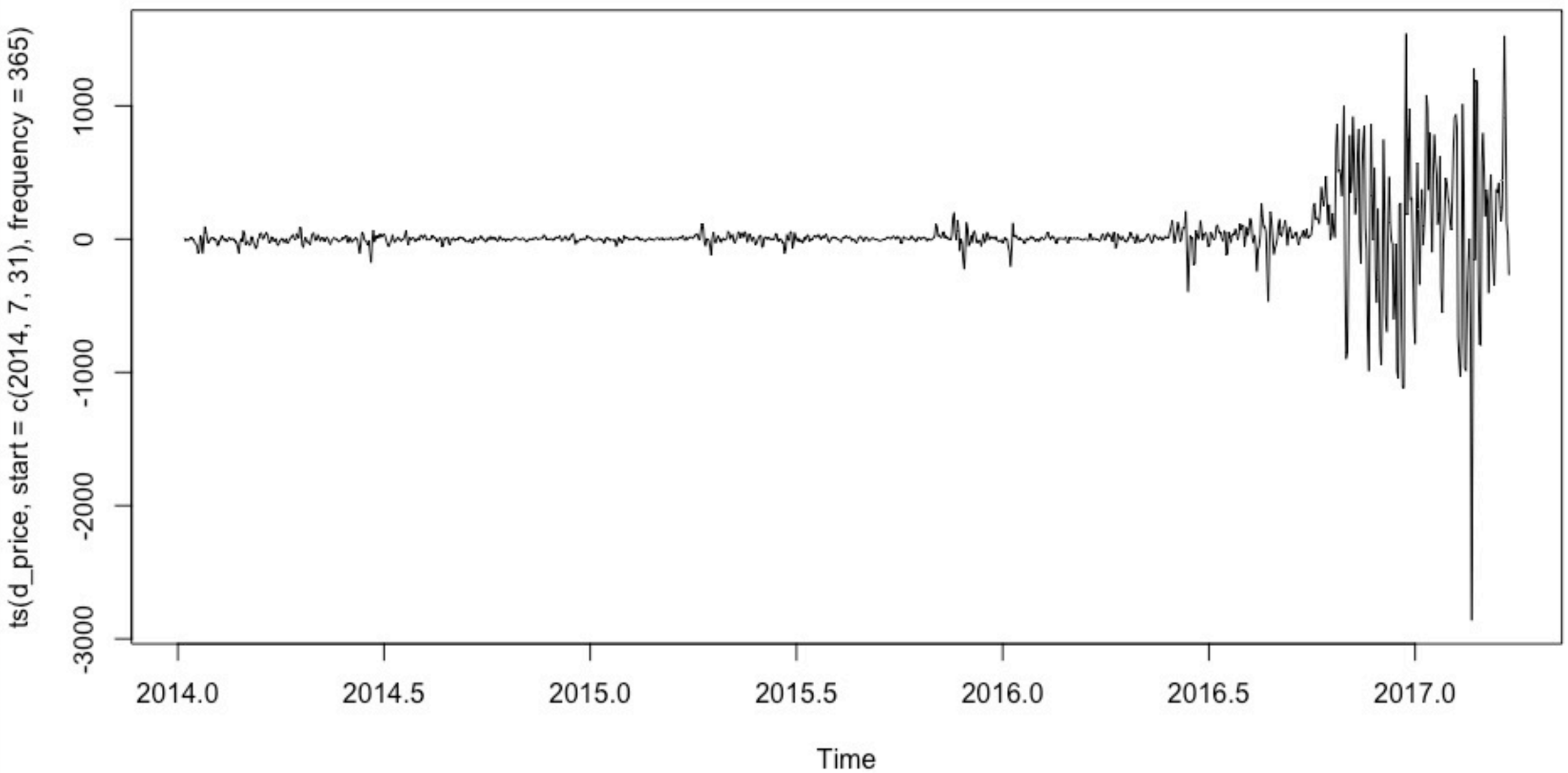

Histogram of price
Density
0.0000
0.0008
0
2000
4000
6000
8000
10000
12000
14000
price


Histogram of price
Density
0
5
15
-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

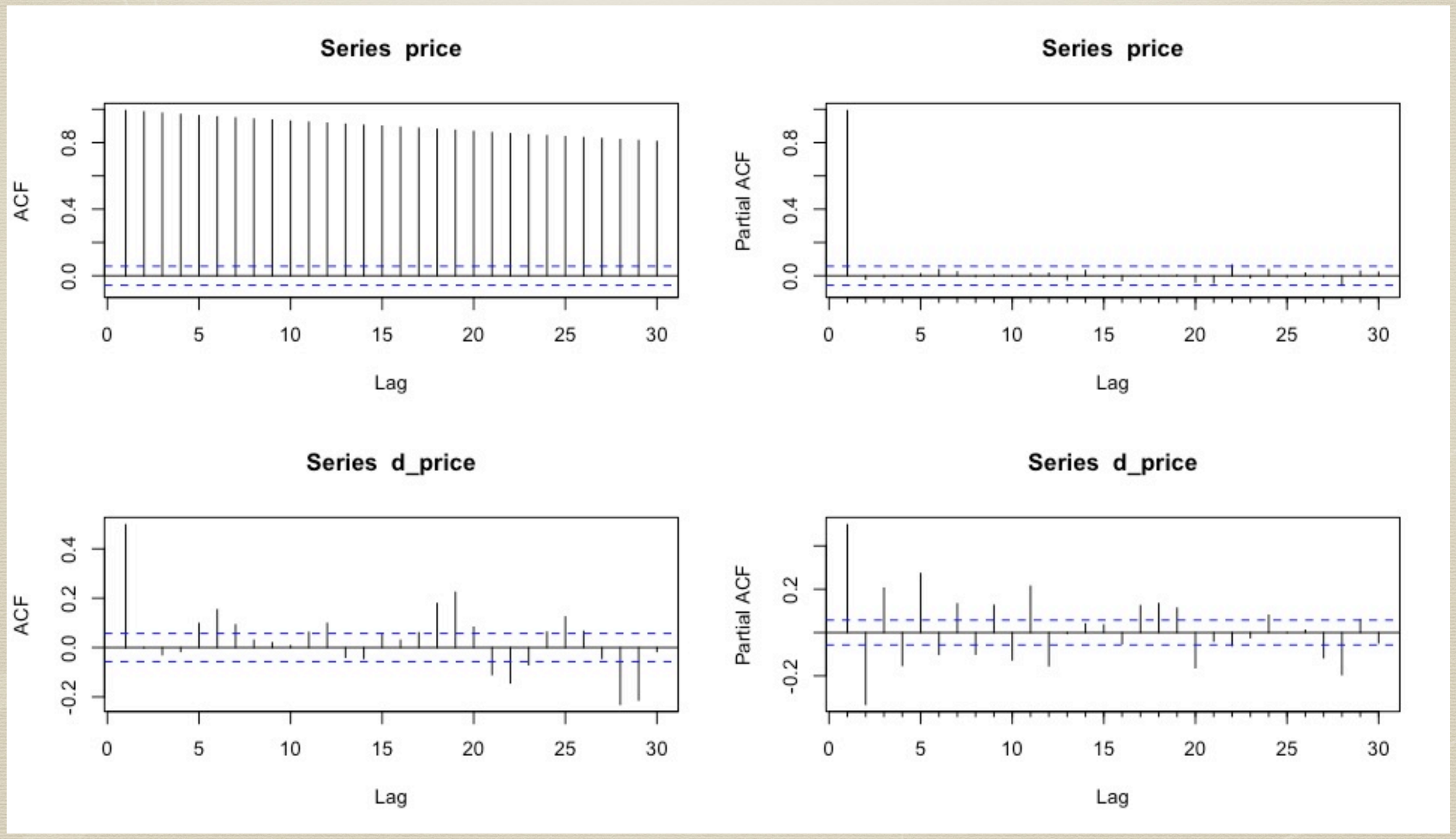
Result

HW4-2 and HW4-3



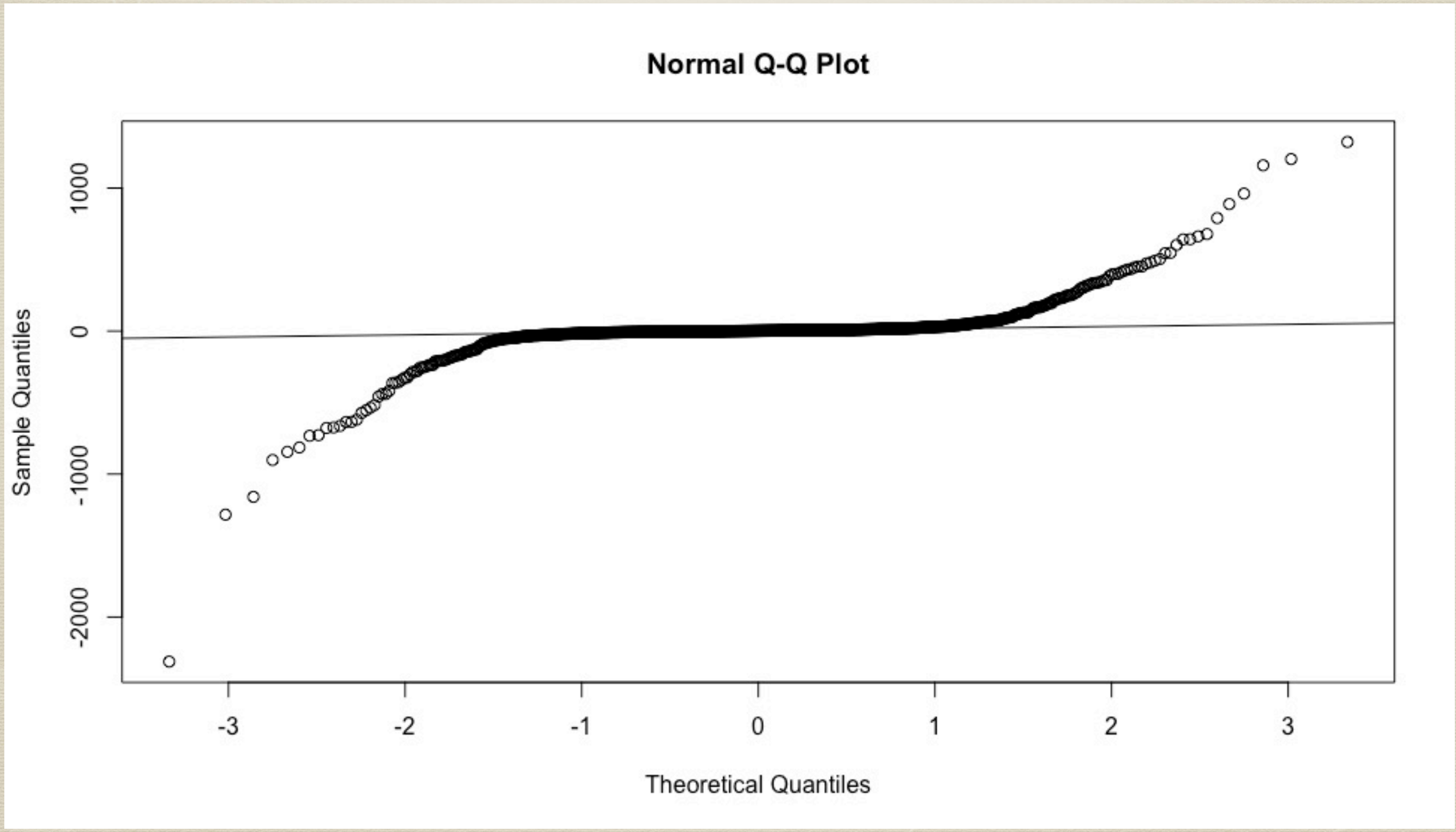
Result

HW4-2 and HW4-3



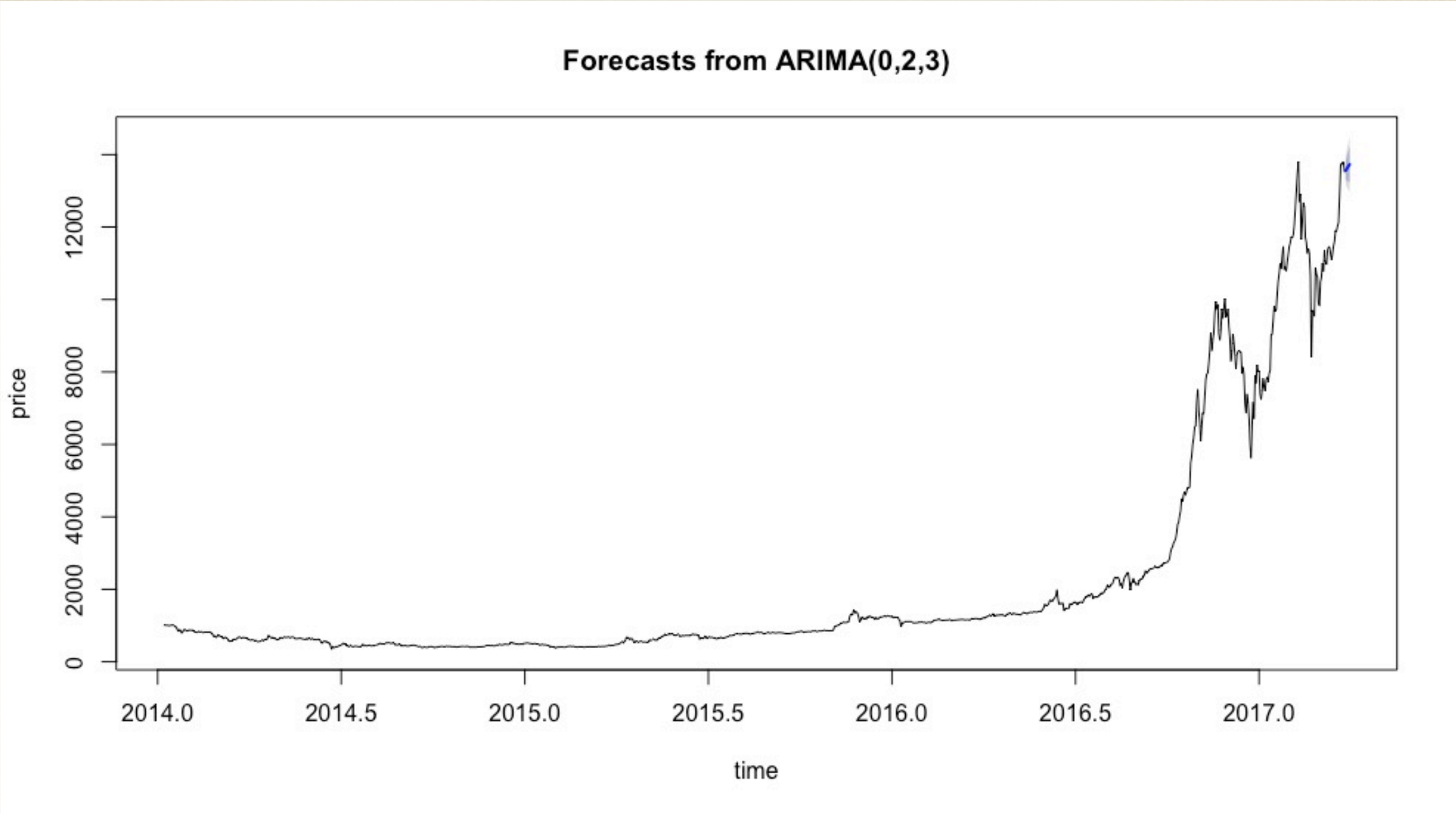
Result

HW4-2 and HW4-3



Result

HW4-2 and HW4-3



Result

HW4-2 and HW4-3

```
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
>
```

```
> forecast(fit,5)
          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
> plot(forecast(fit,5),xlab = "time",ylab = "price")
>
```