**Practical No.1 Estimation and Elimination of trend component various difference method.**

**Q1)**

data()

data("AirPassengers")

AirPassengers

plot(AirPassengers,main="Original AirPassengers data",xlab="Year",ylab="Passengers",type="o")

trend\_estimate\_3yr=filter(AirPassengers,rep(1/36,36),sides=2)

trend\_estimate\_3yr

trend\_estimate\_5yr=filter(AirPassengers,rep(1/60,60),sides=2)

trend\_estimate\_5yr

trend\_eliminate\_3yr=AirPassengers-trend\_estimate\_3yr

trend\_eliminate\_3yr

trend\_eliminate\_5yr=AirPassengers-trend\_estimate\_5yr

trend\_eliminate\_5yr

plot(trend\_estimate\_3yr,main="estimated trend of AirPassenger data",xlab="year",ylab="estimated AirPassengers",type="o")

plot(trend\_estimate\_5yr,main="estimated trend of AirPassenger data",xlab="year",ylab="estimated AirPassengers",type="o")

plot(trend\_eliminate\_3yr,main="eliminate trend of AirPassenger data",xlab="year",ylab="eliminate AirPassengers",type="o")

plot(trend\_eliminate\_5yr,main="eliminate trend of AirPassenger data",xlab="year",ylab="eliminate AirPassengers",type="o")

first\_diff\_series=diff(AirPassengers)

first\_diff\_series

plot(first\_diff\_series,main="differencing series of AirPassenger data",xlab="year",ylab="estimated AirPassengers",type="o")

**Q2)**

data()

data("sunspots")

sunspots

plot(sunspots,main="Original sunspots data",xlab="year",ylab="Observation",type="o")

alpha=0.2

alpha

smoothed\_value=numeric(length(sunspots))

smoothed\_value

for(t in 2:length(sunspots))

{

smoothed\_value[t]=alpha\*sunspots[t-1]+(1-alpha)\*smoothed\_value[t-1]

}

estimated\_trend=smoothed\_value

estimated\_trend

plot(estimated\_trend,main="Estimated trend(Expoential Smoothing)",xlab="Year",ylab="observation")

detrend\_data=sunspots-estimated\_trend

detrend\_data

plot(detrend\_data,main="Detrend data(Trend Estimate)",xlab="Year",ylab="Observation",type="o")

first\_order\_series=diff(sunspots)

first\_order\_series

plot(first\_order\_series,main="monthly sunspots",xlab="year",ylab="Observation",col="red",type="o")

**Practical No.2:Estimation and Elimination of Seasonal component**

**Q1)** data=c(486,474,434,441,435,401,414,414,386,405,411,389,414,426,410,441,459,449,486,510,506,549,579,581,630,666,674,729,771,785)

data

length(data)

plot(data)

plot(data,main="original and smoothed Time Series",ylab="value",xlab="Time",col="blue",type="o")

#it shows their is increasing trend present and no seasonality visible

filter\_coefficient=c(-1,4,3,4,-1)/9

filter\_coefficient

smoothed\_data=filter(data,filter\_coefficient,sides=2)

smoothed\_data

plot(smoothed\_data,main="smoothed Time Series",ylab="value",xlab="Time",type="o",col="red")

mad\_value=mean(abs(data-smoothed\_data),na.rm=TRUE)

mad\_value

msd\_value=mean((data-smoothed\_data)^2,na.rm=TRUE)

msd\_value

cat("mean Square Deviation (MAD):",msd\_value,"/n")

cat("mean Square Deviation (MSD):",mad\_value,"/n")

**Q2)**

data()

original\_data=AirPassengers

original\_data

plot(original\_data,main="Original AirPassengers data",xlab="year",ylab="Passenger",type="o",col="red")

ma\_12=filter(original\_data,rep(1/12,12),sides=2)

ma\_12

plot(ma\_12)

seasonal\_component=original\_data/ma\_12

seasonal\_component

plot(seasonal\_component)

deseasonalized\_data=original\_data/seasonal\_component

deseasonalized\_data

plot(deseasonalized\_data,main="deseasonalized data",xlab="year",ylab="Deseasonalized passengers",col="red",type="o")

diff\_deseasonalized\_data=diff(deseasonalized\_data)

diff\_deseasonalized\_data

plot(diff\_deseasonalized\_data,main="deseasonalized data",xlab="year",ylab="Deseasonalized passengers",col="red",type="o")

**Practical No.3: Examing Stationary . sample ACF and PACF**

#Q1

> library(tseries)

> library(forecast)

> data("LakeHuron")

> LakeHuron

Time Series:

Start = 1875

End = 1972

Frequency = 1

> plot(LakeHuron)

**># The Lke Huron time series shows a fluctuating trend without a clear upward or downword pattern. No seasonality present in the data .**

> L\_mean=mean(LakeHuron);L\_mean

[1] 579.0041

> adf\_test=adf.test(LakeHuron);adf\_test

Augmented Dickey-Fuller Test

data: LakeHuron

Dickey-Fuller = -2.7796, Lag order = 4, p-value = 0.254

alternative hypothesis: stationary

> p\_value1=0.254

> if(p\_value1> 0.05)

+ {

+ print("LakeHuron Time Series is not stationary")

+ }else

+ {

+ print("LakeHuron Time Series is stationary")

+ }

[1] "LakeHuron Time Series is not stationary"

**> # Time series is not stationary so we will use differencing**

> LakeHuron\_diff=diff(LakeHuron);LakeHuron\_diff

Time Series:

Start = 1876

End = 1972

Frequency = 1

> adf\_test\_diff=adf.test(LakeHuron\_diff);adf\_test\_diff

Augmented Dickey-Fuller Test

data: LakeHuron\_diff

Dickey-Fuller = -5.4687, Lag order = 4, p-value = 0.01

alternative hypothesis: stationary

> p\_value2=0.01

> if(p\_value2> 0.05)

+ {

+ print("LakeHuron Time Series is not stationary")

+ }else

+ {

+ print("LakeHuron Time Series is stationary")

+ }

[1] "**LakeHuron Time Series is stationary"**

> acf(LakeHuron\_diff,main="Sample ACF of LakeHuron Time Series")

> pacf(LakeHuron\_diff,main="Sample PACF of LakeHuron Time Series")

#Q2

> library(tseries)

> library(forecast)

> data("BJsales")

> BJsales

Time Series:

Start = 1

End = 150

Frequency = 1

> plot(BJsales)

**># The plot shows a general upward trend with fluctuations, indicating that sales have been increasing over time**

> B\_mean=mean(BJsales);B\_mean

[1] 229.978

> adf\_test=adf.test(BJsales);adf\_test

Augmented Dickey-Fuller Test

Dickey-Fuller = -2.1109, Lag order = 5, p-value = 0.5302

alternative hypothesis: stationary

> p\_value1=0.5302

> if(p\_value1> 0.05)

+ {

+ print("BJsales Time Series is not stationary")

+ }else

+ {

+ print("BJsales Time Series is stationary")

+ }

[1] "BJsales Time Series is not stationary"

**> # Time series is not stationary so we will use differencing.**

> BJsales\_diff=diff(BJsales);BJsales\_diff

Time Series:

Start = 2

End = 150

Frequency = 1

> adf\_test\_diff=adf.test(BJsales\_diff);adf\_test\_diff

Augmented Dickey-Fuller Test

Dickey-Fuller = -3.3485, Lag order = 5, p-value = 0.06585

alternative hypothesis: stationary

> p\_value2= 0.06585

> if(p\_value2> 0.05)

+ {

+ print("BJsales Time Series is not stationary")

+ }else

+ {

+ print("BJsales Time Series is stationary")

+ }

[1] "BJsales Time Series is not stationary"

**> # Time series is still not stationary so again will use differencing.**

> BJsales\_diff2=diff(BJsales\_diff);BJsales\_diff

Time Series:

Start = 2

End = 150

Frequency = 1

> adf\_test\_diff2=adf.test(BJsales\_diff2);adf\_test\_diff2

Augmented Dickey-Fuller Test

Dickey-Fuller = -6.562, Lag order = 5, p-value = 0.01

alternative hypothesis: stationary

> p\_value3= 0.01

> if(p\_value3> 0.05)

+ {

+ print("BJsales Time Series is not stationary")

+ }else

+ {

+ print("BJsales Time Series is stationary")

+ }

**[1] "BJsales Time Series is stationary"**

> acf(BJsales\_diff2,main="Sample ACF of BJsalesTime Series")

> pacf(BJsales\_diff2,main="Sample PACF of BJsales Time Series")

#Q3 > library(tseries)

> library(forecast)

> data("JohnsonJohnson")

> JohnsonJohnson

> plot(JohnsonJohnson)

**># It shows the upward trend ,earning have growb steadily over the years and seasonal fluctuatins , seasonal pattenr in the earnings**.

> J\_mean=mean(JohnsonJohnson);J\_mean

[1] 4.799762

> adf\_test=adf.test(JohnsonJohnson);adf\_test

Dickey-Fuller = 1.9321, Lag order = 4, p-value = 0.99

alternative hypothesis: stationary

> p\_value1=0.99

> if(p\_value1> 0.05)

+ {

+ print("JohnsonJohnson Time Series is not stationary")

+ }else

+ {

+ print("JohnsonJohnson Time Series is stationary")

+ }

[1] "JohnsonJohnson Time Series is not stationary"

**# Time series is not stationary so we will use differencing.**

> JohnsonJohnson\_diff=diff(JohnsonJohnson);JohnsonJohnson\_diff

> adf\_test\_diff=adf.test(JohnsonJohnson\_diff);adf\_test\_diff

Augmented Dickey-Fuller Test

data: JohnsonJohnson\_diff

Dickey-Fuller = -3.9886, Lag order = 4, p-value = 0.01421

alternative hypothesis: stationary

> p\_value2= 0.01421

> if(p\_value2> 0.05)

+ {

+ print("JohnsonJohnson Time Series is not stationary")

+ }else

+ {

+ print("JohnsonJohnson Time Series is stationary")

+ }

**[1] "JohnsonJohnson Time Series is stationary"**

> acf(JohnsonJohnson\_diff,main="Sample ACF of JohnsonJohnson Time Series")

> pacf(JohnsonJohnson\_diff,main="Sample PACF of JohnsonJohnson Time Series")

#Q4> library(tseries)

> library(forecast)

> data("AirPassengers")

> AirPassengers

> plot(AirPassengers)

**>#The plot shows an increasing trend over time, indicating a rise in the number of passengers. Additionally, there is a clear seasonal pattern,**

> A\_mean=mean(AirPassengers);A\_mean

[1] 280.2986

> adf\_test=adf.test(AirPassengers);adf\_test

Augmented Dickey-Fuller Test

Dickey-Fuller = -7.3186, Lag order = 5, p-value = 0.01

alternative hypothesis: stationary

> p\_value=0.01

> if(p\_value> 0.05)

+ {

+ print("AirPassengers Time Series is not stationary")

+ }else

+ {

+ print("AirPassengers Time Series is stationary")

+ }

[1] "AirPassengers Time Series is stationary"

> acf(AirPassengers,main="Sample ACF of AirPassengers Time Series")

> pacf(AirPassengers,main="Sample PACF of AirPassengers Time Series")

**Practical No.4:identification of MA and AR process and its order selection**

**Q1)**

library(tseries)

library(forecast)

data=read.csv("C:\\Users\\DELL\\Desktop\\pollution.csv")

data

head(data)

T\_data=data[,'pm2.5']

T\_data

head(T\_data)

T\_data=na.omit(T\_data)

T\_data

view(T\_data)

head(T\_data)

plot(T\_data)

adf\_test=adf.test(T\_data)

adf\_test

acf(T\_data,main="Autocorrelation Function(ACF)")

pacf(T\_data,main="Partial Autocorrelation Function(ACF)")

T\_data\_diff=diff(T\_data)

T\_data\_diff

head(T\_data\_diff)

acf(T\_data\_diff,main="Autocorrelation Function(ACF)")

pacf(T\_data\_diff,main=" Partial Autocorrelation Function(ACF)")

ar\_model=ar(T\_data\_diff,order=4)

ar\_model

ma\_model=ma(T\_data\_diff,order=2)

ma\_model

head(ma\_model)

ar\_forecast=forecast(ar\_model,h=12)

ar\_forecast

ma\_forecast=forecast(ma\_model,h=12)

ma\_forecast

plot(ma\_forecast)

plot(ar\_forecast)

**Practical No.5:Yule Walker Equation for AR(p) Model**

**Q1)**

data()

data(AirPassengers)

AirPassengers

plot(AirPassengers)

acf\_values=acf(AirPassengers,plot=FALSE)

acf\_values

acf\_vals=acf\_values$acf

acf\_vals

gamma\_o=acf\_vals[1,1,1]

gamma\_o

gamma\_1=acf\_vals[2,1,1]

gamma\_1

gamma\_2=acf\_vals[3,1,1]

gamma\_2

Yule\_walker\_matrix=matrix(c(gamma\_o,gamma\_1,gamma\_2),nrow=2,byrow=TRUE)

Yule\_walker\_matrix

R1=c(gamma\_1,gamma\_2)

R1

Ar=solve(Yule\_walker\_matrix,R1)

Ar

**Practical No.6:Fitting MA Model using Least Square Regression**

**Q2)**

library(tseries)

data()

data(sunspot.month)

sunspot.month

plot(sunspot.month,main="Monthly sunspots Data",ylab="Sunspots Number",xlab="Time")

adf\_test=adf.test(sunspot.month)

adf\_test

ar\_model=ar(sunspot.month,order.max=2,method="yw")

ar\_model

resid\_est=ar\_model$resid

resid\_est

n=length(resid\_est)

n

y=sunspot.month[3:n]

y

e1=resid\_est[1:(n-1)]

e1

e2=resid\_est[2:(n-2)]

e2

head(e2)

ma2\_model=lm(y~e1+e2)

ma2\_model

summary(ma2\_model)

**Practical No.7: Residual Analysis and Diagnostic Checking**

**Q1)**

library(tseries)

library(forecast)

data()

data(AirPassengers)

AirPassengers

plot(AirPassengers,main="AirPassengers Dataset",ylab="No of Passengers",xlab="Year")

fit=auto.arima(AirPassengers)

fit

residuals=residuals(fit)

residuals

head(residuals)

plot(residuals,main="residuals form fitted",xlab="year",ylab="residuals")

acf(residuals,main="acf of residuals")

ljung\_box\_test=Box.test(residuals,lag=20,type="Ljung-Box")

ljung\_box\_test

print(ljung\_box\_test)

if(ljung\_box\_test$p.value>0.05)

{print("Residual are independent, model is appropriate")}

else

{print("Residual are autocorrelated, model might need improvement")}

**Practical No.8: Fitting ARMA Model**

**Q1)**

data=read.csv("C:\\Users\\DELL\\Desktop\\Amazon.csv")

data

head(data)

library(tseries)

library(forecast)

data=data[,"rt"]

data

plot(data)

adf\_test=adf.test(data)

adf\_test

acf=acf(data,main="ACF of amazon data")

pacf=pacf(data,main="PACF of amazon data")

p=1

q=1

model=arima(data,order=c(p,0,q))

model

bic\_value=BIC(model)

bic\_value

aic\_value=AIC(model)

aic\_value

aicc\_value=AIC(model,k=log(length(data)))

aicc\_value

**Q2)**

data=read.csv("C:\\Users\\DELL\\Desktop\\Gold.csv")

data

head(data)

library(tseries)

library(forecast)

data=data[,"VALUE"]

data

plot(data)

adf\_test=adf.test(data)

adf\_test

data\_diff=diff(data)

data\_diff

adf\_test=adf.test(data\_diff)

adf\_test

acf=acf(data,main="ACF of Gold data")

pacf=pacf(data,main="PACF of Gold data")

p=1

q=1

model=arima(data,order=c(p,0,q))

model

bic\_value=BIC(model)

bic\_value

aic\_value=AIC(model)

aic\_value

aicc\_value=AIC(model,k=log(length(data)))

aicc\_value

**Practical No: 9**

**Dickey Fuller Unit Root Test**

**##Q1**

> plot(data)

> data=read.csv("C:\\Users\\hp\\Downloads\\monthly-housing.csv")

> data

> library(tseries)

> cat("Ho:The time series has a unit root. i.e it is non stationary")

Ho:The time series has a unit root. i.e it is non stationary> cat("H1:The time series does not have a unit root")

H1:The time series does not have a unit root> data=na.omit(data)

> data

> plot(data)

>

data\_hpi=data[,"hpi"]

> data\_hpi

[##There is a increasing trend in data.

data\_nsold=data[,"numsold"]

> data\_nsold

##There is increasing trend initially and then decreases suddenly.

adf\_test\_hpi=adf.test(data\_hpi)

> print(adf\_test\_hpi) ##Here p value is greater than 0.05 so series is non stationary

> p\_value=0.8833

> if(p\_value>0.05)

+ {

+ print("Time series is not stationary")

+ }else

+ {

+ print("Time series is statinary")

+ }

[1] "Time series is not stationary"

> adf\_test\_nsold=adf.test(data\_nsold)

> print(adf\_test\_nsold) ##Here p value is greater than 0.05 so series is non stationary

> if(p\_value>0.05)

+ {

+ print("Time series is not stationary")

+ }else

+ {

+ print("Time series is statinary")

+ }

[1] "Time series is not stationary"

##Q2.

> data=read.csv("C:\\Users\\hp\\Downloads\\Gold.csv")

> data

> cat("Ho:The time series has a unit root. i.e it is non stationary")

Ho:The time series has a unit root. i.e it is non stationary> cat("H1:The time series does not have a unit root")

H1:The time series does not have a unit root> data\_value=data[,"VALUE"]

> data\_value

[1]

> plot(data\_value)

## Initially upto 2000 data is gradually increases then after 2000 data is sudden increases. And we see there is upward trend in data.

> adf\_test=adf.test(data\_value)

> adf\_test

> p\_value=0.6654

> if(p\_value>0.05)

+ {

+ print("Time series is not stationary")

+ }else

+ {

+ print("Time series is statinary")

+ }

[1] "Time series is not stationary"

>

**Practical No.10: Identification of ARIMA(p,d,q) Process and order selection.**

**Q1)**

library(forecast)

data=read.csv("C:\\Users\\DELL\\Desktop\\Gold.csv")

data

T\_data=data[,"VALUE"]

T\_data

plot(T\_data)

gold\_diff=(T\_data)

gold\_diff

adf\_test=adf.test(gold\_diff)

adf\_test

print(adf\_test\_diff)

acf(gold\_diff,main="ACF of Diff Series")

pacf(gold\_diff,main="PACF of Diff Series")

arima=arima(gold\_diff,order=c(1,1,1))

arima

**Practical No.11: select a series and obtain mean ,variance and auto covariance autocorrelation upto lag 5.**

**Q1)**

data=c(47,64,23,71,38,64,55,41,59,48,71,35,57,40,58,44,80,55,37,74,51,57,50,60,45,57,50,45,25,59,50,71,56,74,58,58,45,54,36,54,48,55,45,57,50,62,44,64,43,52,38,59,55,41,53,49,34,35,54,45,68,38,50,60,39,59,40,57,54,23)

data

length(data)

m=mean(data)

m

v=var(data)

v

auto=acf(data,lag=5,plot=T)

auto

data\_ts=ts(data)

data\_ts

acvf\_result=acf(data\_ts,lag.max=5,type="covariance",plot=T)

acvf\_result

acvf\_result$acf

**Q2)**

data1=read.csv("C:\\Users\\DELL\\Desktop\\Gold.csv")

data1

head(data1)

m=mean(data1)

m

v=var(data1)

v

auto=acf(data1,lag=50,plot=T)

auto

data\_ts=ts(data)

data\_ts

acvf\_result=acf(data\_ts,lag.max=50,type="covariance",plot=T)

acvf\_result

acvf\_result$acf

**Practical No.13:Stratified random sample(Various type of allocation method**)

**Q1)**

x1=c(797,773,748,734,588,577,507,507,457,438,415,401,387,381,324,315)

x1

N1=length(x1)

N1

x2=c(314,298,296,258,256,243,238,237,172,172,163,162,161,159,153,144,121,120,119,118,118,116,116,113,235,235,216,208,201,192,180,179,138,138,138,138,136,132,130,126,113,110,110,108,106,104,101,100)

x2

N2=length(x2)

N2

X=c(x1,x2)

X

length(X)

N=length(X)

N

n=24

s1=sample(X,n,replace=TRUE)

s1

m1=mean(s1)

m1

S1=sum((s1-m1)^2)/(n-1)

S1

SD1=sqrt(((N-1)\*S1)/(N\*n))

SD1

#Sample for SRSWOR

s2=sample(X,n,replace=FALSE)

s2

m2=mean(s2)

m2

S2=sum((s2-m2)^2)/(n-1)

S2

SD2=sqrt(((N-1)\*S2)/(N\*n))

SD2

n1=(n/N)\*N1

n1

n2=(n/N)\*N2

n2

W1=N1/N

W1

W2=N2/N

W2

s2=sample(x2,n2,TRUE)

s2

s=c(s1,s2)

s

m3=mean(m1)

m3

m4=mean(m2)

m4

Sq1=sum((s1-m3)^2)(n1-1)

Sq1

Sq2=sum((s2-m4)^2)(n2-1)

Sq2

SD=sqrt(((1/n)-)(1/N))\*(W1\*Sq1+W2\*Sq2))

SD

#neymann allocation

t1=var(x1)

t1

t2=var(x2)

t2

n1=(W1\*sqrt(t1)/(W1\*sqrt(t1)+W2\*sqrt(t2)))\*n

n1

n2=(W2\*sqrt(t2)/(W1\*sqrt(t1)+W2\*sqrt(t2)))\*n

n2

S1=sample(x1,n1,FALSE)

S1

S1=sample(x2,n2,FALSE)

S1

Si1=sum((S1-mean(S1))^2)/(n1-1)

Si1

Si2=sum((S2-mean(S2))^2)/(n2-1)

Si2

sd=sqrt((((W!\*sqrt(Si1))^+(W2\*sqrt(Si2))^2)/n)-(W1\*Si1+W2\*Si2)/N))

sd

**Practical no.14: sratified random sampling and regression method of estimation**

**Q1)**

x=c(1054,973,1089,1054,894)

x

y=c(10316,7025,10512,8963,8783)

y

n=5

n

xbar=mean(x)

xbar

ybar=mean(y)

ybar

Rn=ybar/xbar

Rn

YR=Rn\*xbar

YR

s2x=var(x)

s2x

s2y=var(y)

s2y

sxy=cov(x,y)

sxy

S=s2y+Rn^2\*s2x-2\*Rn\*sxy

S

SE=sqrt(((1/n)-(1/N))\*S)

SE

beta=sxy/s2x

beta

P=s2y+beta^2\*s2x-2\*beta\*sxy

p

se=sqrt(((1/n)-(1/N)\*p))

se

Xbar=988.75

Yd=ybar+beta\*(Xbar-xbar)

Yd

eff=Yd/YR

eff

**Q2)**

x=c(61,42,50,58,67,45,39,57,71,53)

x

y=c(59,47,52,60,67,48,44,58,76,58)

y

n=10

n

N=200

N

xbar=mean(x)

xbar

ybar=mean(y)

ybar

Rn=ybar/xbar

Rn

YR=Rn\*xbar

YR

s2x=var(x)

s2x

s2y=var(y)

s2y

sxy=cov(x,y)

sxy

S=s2y+Rn^2\*s2x-2\*Rn\*sxy

S

SE=sqrt(((1/n)-(1/N))\*S)

SE

beta=sxy/s2x

beta

P=s2y+beta^2\*s2x-2\*beta\*sxy

p

se=sqrt(((1/n)-(1/N))\*p)

se

Xbar=11600/200

Xbar

Yd=ybar+beta\*(Xbar-xbar)

Yd

eff=Yd/YR

eff

**practical No.15: Circular systematic Sampling**

**Q1)**

>x=c(26,28,11,16,07,22,44,26,31,26,16,9,22,26,17,39,21,14,40,30,27,20,25,39,24,25,18,44,5

5,39,37,14,14,24,18,17,14,38,36,29,04,05,11,09,25,16,13,22,18,06,36,20,43,27,20,21,18,19,2

4,30,20,21,15,14,13,09,25,17,07,30,21,26,16,18,11,19,27,29)

* N=length(x); N

[1] 78

* n=8
* k=floor(N / n); k

[1] 9

* y=seq(1, N); y

[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29

[30] 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57

58

[59] 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78

* Z=matrix(0, nrow=n, ncol=N); Z
* for (j in 1:N) {

+ r[j] <- sample(y, 1)

+ for (i in 1:n) {

+ idx <- r[j] + i \* k

+ if (idx <= N) {

+ Z[i, j] <- idx

+ } else {

+ Z[i, j] <- idx - N

+ }

+ }

+ }

* ​
* sample=matrix(x[Z], nrow=n, ncol=N);sample
* mean=colMeans(sample);mean

>var=var(col\_means); var

[1] 7.620528

* s=sample(x, 8, replace=FALSE); s

[1] 29 30 17 44 21 21 18 30

* m=mean(s); m

[1] 26.25

* v=var(s); v

[1] 79.92857

**Practical nNo. 16 : cluster sampling with equal and unequal sample**

**Q1)**

>x=c(26,28,11,16,07,22,44,26,31,26,16,9,22,26,17,39,21,14,40,30,27,20,25,39,24,25,18,44,5

5,39,37,14,14,24,18,17,14,38,36,29,04,05,11,09,25,16,13,22,18,06,36,20,43,27,20,21,18,19,2

4,30,20,21,15,14,13,09,25,17,07,30,21,26,16,18,11,19,27,29)

* N=length(x); N

[1] 78

* n=8
* k=floor(N / n); k

[1] 9

* y=seq(1, N); y

[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29

[30] 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57

58

[59] 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78

* Z=matrix(0, nrow=n, ncol=N); Z
* for (j in 1:N) {

+ r[j] <- sample(y, 1)

+ for (i in 1:n) {

+ idx <- r[j] + i \* k

+ if (idx <= N) {

+ Z[i, j] <- idx

+ } else {

+ Z[i, j] <- idx - N

+ }

+ }

+ }

* ​
* sample=matrix(x[Z], nrow=n, ncol=N);sample
* mean=colMeans(sample);mean

[1] 21.000 24.375 23.250 19.000 21.750 19.125 21.375 27.000 26.750 25.125 19.625 23.875

[13] 21.750 21.750 21.375 26.500 23.000 26.500 22.125 23.750 23.625 22.750 22.500

21.750

[25] 27.875 17.875 23.250 25.750 19.625 18.750 22.375 17.125 22.125 27.875 16.000

22.625

[37] 21.750 22.375 18.750 28.750 15.500 22.000 25.625 22.500 26.500 21.375 27.125

27.125

[49] 23.750 22.500 20.875 23.875 21.625 19.625 28.750 31.500 14.125 23.875 24.250

27.125

[61] 23.125 27.750 21.625 17.250 23.250 22.375 23.125 24.625 26.500 27.500 18.750

24.250

[73] 27.125 23.000 21.375 17.875 26.750 25.625

* var=var(col\_means); var

[1] 7.620528

* s=sample(x, 8, replace=FALSE); s

[1] 29 30 17 44 21 21 18 30

* m=mean(s); m

[1] 26.25

* v=var(s); v

[1] 79.92857

# Practical No. 17-Jacknife and bootstrap method

**Question 1**

>x=c(8,26,6.33,10.4,5.27,5.35,5.61,6.12,6.

19,5.2,7.01,8.74,7.78,7.01,6,6.5,8,5.12,7.4

1,6.52,6.21,12.28,5.6,5.38,6.6,8.74)

* x

[1] 8.00 26.00 6.33 10.40 5.27 5.35

5.61 6.12 6.19 5.20 7.01 8.74 7.78

7.01

[15] 6.00 6.50 8.00 5.12 7.41 6.52

6.21 12.28 5.60 5.38 6.60 8.74

* original\_cv=(sd(x)/mean(x))\*100
* original\_cv

[1] 53.50442

* n\_bootstrap=1000
* n\_bootstrap

[1] 1000

* bootstrap\_means=numeric(n\_bootstrap)
* bootstrap\_vars=numeric(n\_bootstrap)
* bootstrap\_cvs=numeric(n\_bootstrap)
* for(i in 1:n\_bootstrap)

+ {

+

bootstrap\_sample=sample(x,size=length(x)

,replace=TRUE)

+

bootstrap\_means[i]=mean(bootstrap\_samp le)

+ bootstrap\_vars[i]=var(bootstrap\_sample)

+

bootstrap\_cvs[i]=(sd(bootstrap\_sample)/m ean(bootstrap\_sample))\*100

+

* #bootstrap\_means
* #bootstrap\_vars
* #bootstrap\_cvs
* hist(bootstrap\_cvs)

>lowerquartile=quantile(bootstrap\_cvs,0.0 25)

* lowerquartile 2.5%

17.0763

>upperquartile=quantile(bootstrap\_cvs,0.0 95)

* upperquartile 9.5%

20.95215

* bias\_estimate=mean(bootstrap\_cvs)- original\_cv

bias\_estimate

[1] -8.174178

* corrected\_cv=original\_cv-bias\_estimate
* corrected\_cv

[1] 61.6786

# Question 2

* x=c(24,26,32,36,43,52,62,56,52,21)
* y=c(22,28,5,18,14,14,8,8,10,24)
* original\_corr=cor(x,y)
* original\_corr

[1] -0.7284006

* n=length(x)
* n

[1] 10

* jackknife\_corr=numeric(n)
* jackknife\_corr

[1] 0 0 0 0 0 0 0 0 0 0

* for(i in 1:n)

+ {

+ x\_jackknife=x[-i]

+ y\_jackknife=y[-i]

+

jackknife\_corr[i]=cor(x\_jackknife,y\_jackk nife)

+ }

* jackknife\_corr

[1] -0.6927381 -0.6920750 -0.9451279 -

0.7248160 -0.7276008 -0.7447716 -

0.6926761

[8] -0.6920055 -0.7104119 -0.6667885

* ​

jackknife\_estimator=mean(jackknife\_corr)

* jackknife\_estimator

[1] -0.7289011

* bias=jackknife\_estimator-jackknife\_corr
* bias

[1] -0.036163078 -0.036826127

0.216226749 -0.004085145 -0.001300329

0.015870459

[7] -0.036225077 -0.036895587 -

0.018489236 -0.062112630

* sd=sqrt(((n-1)/n)\*sum((jackknife\_corr- jackknife\_estimator)^2))
* sd

[1] 0.2256223

# Question 3

* x=c(22,26,58,54,30,35,12,28)
* x

[1] 22 26 58 54 30 35 12 28

* n=length(x)
* n

[1] 8

* m=mean(x)
* m

[1] 33.125

* u2=sum((x-m)^2)/n
* u2

[1] 214.3594

* u3=sum((x-m)^3)/n
* u3

[1] 1645.488

* beta=(u3^2)/(u2^3)
* beta

[1] 0.274892

* gamma=sqrt(beta)
* gamma

[1] 0.5243015

* n\_bootstrap=8
* n\_bootstrap

[1] 8

* m1=numeric(n\_bootstrap)
* m1

[1] 0 0 0 0 0 0 0 0

* mu2=numeric(n\_bootstrap)
* mu2

[1] 0 0 0 0 0 0 0 0

* mu3=numeric(n\_bootstrap)
* mu3

[1] 0 0 0 0 0 0 0 0

* b1=numeric(n\_bootstrap)
* b1

[1] 0 0 0 0 0 0 0 0

* g1=numeric(n\_bootstrap)
* g1

[1] 0 0 0 0 0 0 0 0

* for(i in 1:n\_bootstrap)

+ {

+

bootstrap\_sample=sample(x,size=length(x)

,replace=TRUE)

+ m1[i]=mean(bootstrap\_sample)

+ mu2[i]=sum((bootstrap\_sample- m1[i])^2)/n

+ mu3[i]=sum((bootstrap\_sample- m1[i])^3)/n

+ b1[i]=(mu3[i]^2)/(mu2[i]^3)

+ g1[i]=sqrt(b1[i])

+ }

* m1

[1] 33.375 32.875 33.625 37.125 35.625

34.000 29.125 29.875

* mu2

[1] 180.2344 140.1094 181.9844 269.3594

222.2344 210.7500 127.3594 124.6094

* mu3

[1] 863.3086 828.6680 2308.9102

318.3633 813.6914 1109.2500 1268.6133

995.3555

* b1

[1] 0.127297318 0.249666065

0.884528364 0.005186205 0.060323391

0.131448648 0.779050750

[8] 0.512040438

* g1

[1] 0.35678750 0.49966595 0.94049368

0.07201531 0.24560821 0.36255848

0.88263852

[8] 0.71557001

* mg1=mean(g1)
* mg1

[1] 0.5094172

* bias=mg1-gamma
* bias

[1] 0.05180846

* se\_gamma=sd(g1)
* se\_gamma

[1] 0.362564

# Question 4

* x=c(32,4,16,7,12,27)
* y=c(2300,30,1500,15,700,1800)
* mx=mean(x)
* mx

[1] 16.33333

* my=mean(y)
* my

[1] 1057.5

* n\_bootstrap=6
* n\_bootstrap

[1] 6

* m1=numeric(n\_bootstrap)
* m1

[1] 0 0 0 0 0 0

* m2=numeric(n\_bootstrap)
* m2

[1] 0 0 0 0 0 0

* m3=numeric(n\_bootstrap)
* m3

[1] 0 0 0 0 0 0

* m4=numeric(n\_bootstrap)
* m4

[1] 0 0 0 0 0 0

* for(i in 1:n\_bootstrap

+

bootstrap\_sample1=sample(x,size=length( x),replace=TRUE)

+

bootstrap\_sample2=sample(y,size=length( y),replace=TRUE)

+ m1[i]=mean(bootstrap\_sample1)

+ m2[i]=mean(bootstrap\_sample2)

+ }

* m1

[1] 20.16667 23.50000 24.83333 14.50000

21.16667 19.00000

* m2

[1] 1326.6667 985.8333 1035.8333

729.1667 1519.1667 607.5000

* for(i in 1:n\_bootstrap)

+ {

+ x\_jackknife=x[-i]

+ y\_jackknife=y[-i]

+ m3[i]=mean(x\_jackknife)

+ m4[i]=mean(y\_jackknife)

+ }

* m3

[1] 13.2 18.8 16.4 18.2 17.2 14.2

* m4

[1] 809 1263 969 1266 1129 909

* mean(m1)-mx

[1] 4.194444

**Practical No.19: Probability Proportional to size (pps) sampling**

# Question 1)

> n=c(1,2,3,4,5,6,7,8,9,10)

* n

[1] 1 2 3 4 5 6 7 8 9 10

> trees=c(150,50,80,100,200,160,40,220,60,140)

* trees

[1] 150 50 80 100 200 160 40 220 60 140

* c=c()
* c[1]=trees[1]
* for(i in 2:length(trees))

+ {

+ trees[i]=trees[i-1]+trees[i]

+ c[i]=trees[i]

+ }

* c

[1] 150 200 280 380 580 740 780 1000

1060 1200

* ns=4
* ns

[1] 4

* sn=numeric(ns)
* sn

[1] 0 0 0 0

* s=numeric(ns)
* s

[1] 0 0 0 0

* for(j in 1:ns)

+ {

+ sn[j]=sample(1:c[length(c)],1)

+ sample\_index=which(c>=sn[j])[1]

+ s[j]=sample\_index

+ }

* sn

[1] 1036 810 458 895

* s

[1] 9 8 5 8

# Question 2)

* n=10
* n

[1] 10

* N=800
* N

[1] 800

* x=c(5511,865,2535,3523,8368,7357,5131,465 4,1146,1165)
* x

[1] 5511 865 2535 3523 8368 7357 5131

4654 1146 1165

* y=c(4824,924,1948,3013,7378,5506,4051,406 0,809,1013)
* y

[1] 4824 924 1948 3013 7378 5506 4051

4060 809 1013

* Total\_population=415149
* Total\_population

[1] 415149

* p=x/sum(x)
* p

[1] 0.13690225 0.02148801 0.06297354

0.08751708 0.20787480 0.18275991

0.12746243

[8] 0.11561297 0.02846851 0.02894050

* Ty=(N/n)\*sum(y/p)
* Ty

[1] 27162999

* var=(N/n)\*sum((y/p-y)^2\*p)
* var

[1] 67496918546

* SE=sqrt(var)
* SE

[1] 259801.7

# Question 3)

* n=c(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15)
* n

[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

* nre=c(348.334,3.433,431.439,848.317,3928.73 2,906.281,4.373,43.229,464.516,540.696,38.0

67,1006.036,2610.572,1022.782,3909.738)

* nre

[1] 348.334 3.433 431.439 848.317

3928.732 906.281 4.373 43.229 464.516

[10] 540.696 38.067 1006.036 2610.572

1022.782 3909.738

* re=c(409,2.605,54.633,907.7,1343.461,315.80 9,7.13,42.808,825.748,939.46,40.775,53.753,2

131.048,1213.024,2327.025)

* re

[1] 409.000 2.605 54.633 907.700

1343.461 315.809 7.130 42.808 825.748

[10] 939.460 40.775 53.753 2131.048

1213.024 2327.025

* ns=50
* i=sample(n,ns,replace=T)
* i

[1] 2 9 12 6 9 10 11 10 7 12 10 11 3 2 4

3 11 12 1 13 5 7 9 10 6 1 12 13

[29] 15 11 2 13 12 12 10 11 9 10 2 1 4 4 12

13 9 14 12 12 7 1

* n1=seq(1:max(nre))
* j=sample(n1,ns,replace=T)
* j

[1] 77 1058 622 662 2272 1315 207 3432

1644 2714 2898 1026 2739 106 1812 2191

492

[18] 176 808 3161 1584 1556 3846 550

1696 3467 2270 3837 1261 2288 1595 649

214 1735

[35] 2954 2502 2397 3662 1461 38 3771

3314 219 2670 547 3728 413 549 3223

3051

* x=nre[i]
* x

[1] 3.433 464.516 1006.036 906.281

464.516 540.696 38.067 540.696 4.373

[10] 1006.036 540.696 38.067 431.439

3.433 848.317 431.439 38.067 1006.036

[19] 348.334 2610.572 3928.732 4.373

464.516 540.696 906.281 348.334 1006.036

[28] 2610.572 3909.738 38.067 3.433

2610.572 1006.036 1006.036 540.696

38.067

[37] 464.516 540.696 3.433 348.334

848.317 848.317 1006.036 2610.572 464.516

[46] 1022.782 1006.036 1006.036 4.373

348.334

* sam=numeric(ns)
* for(k in 1:ns)

+ {

+ if(i[k]<=x[k] && j[k]<=x[k])

+ {

+ sam[k]=i[k]

+ }

+ }sam

[1] 0 0 12 6 0 0 0 0 0 0 0 0 0 0 0 0 0

12 0 0 5 0 0 0 0 0 0 0

[29] 15 0 0 13 12 0 0 0 0 0 0 1 0 0 12 0

0 0 12 12 0 0