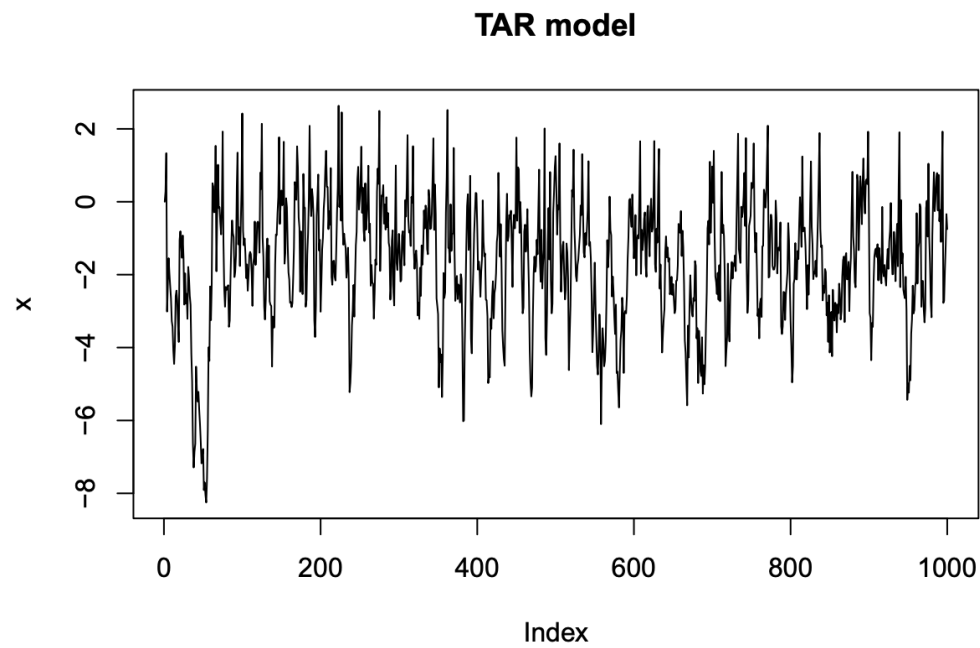
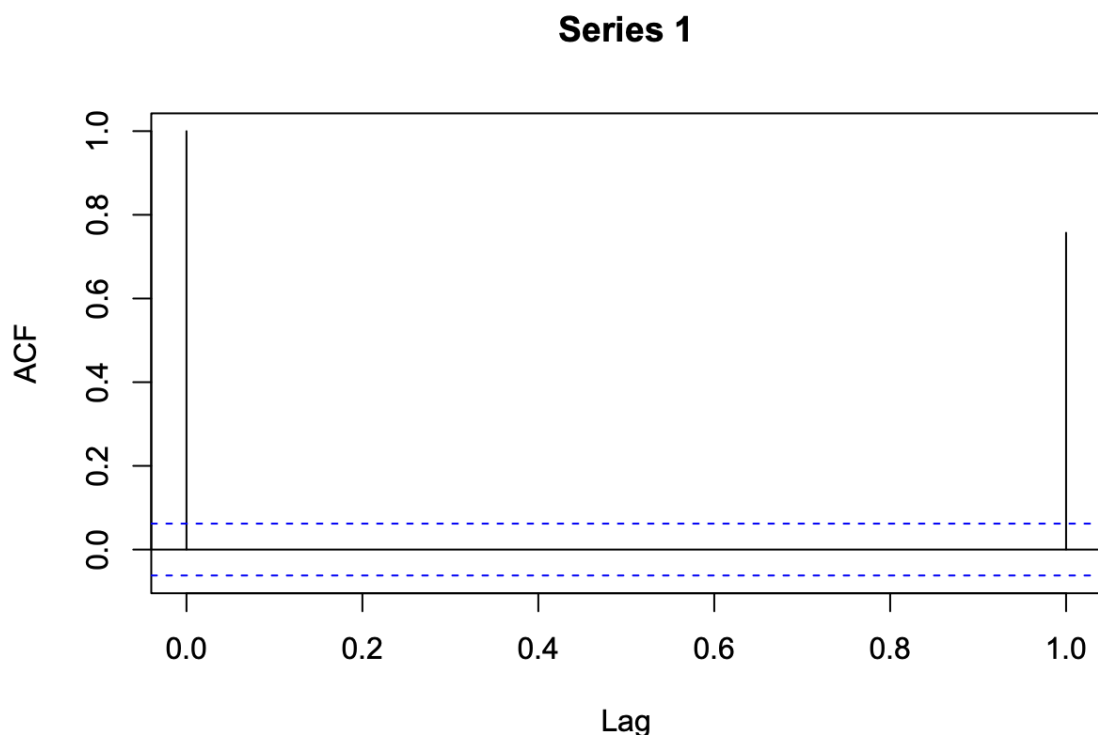


Question 1



ACF plot



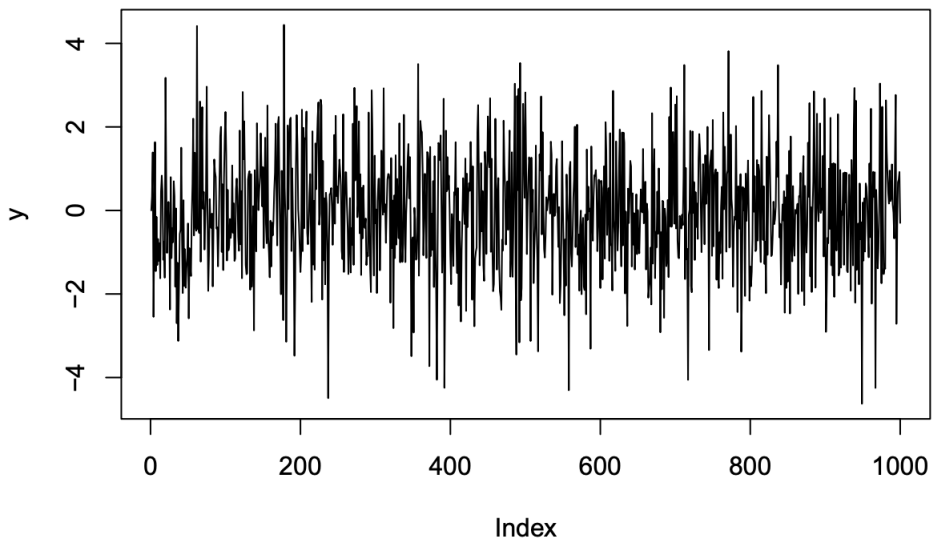
Mean and Std. dev:

Mean: -1.642559

STD.Deviation: 1.710616

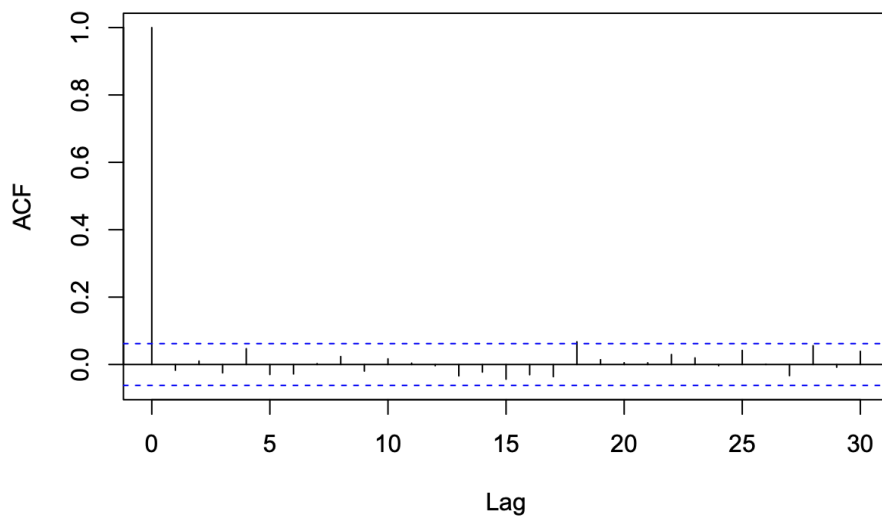
Question 2

TFN model



ACF Plot

Series 1



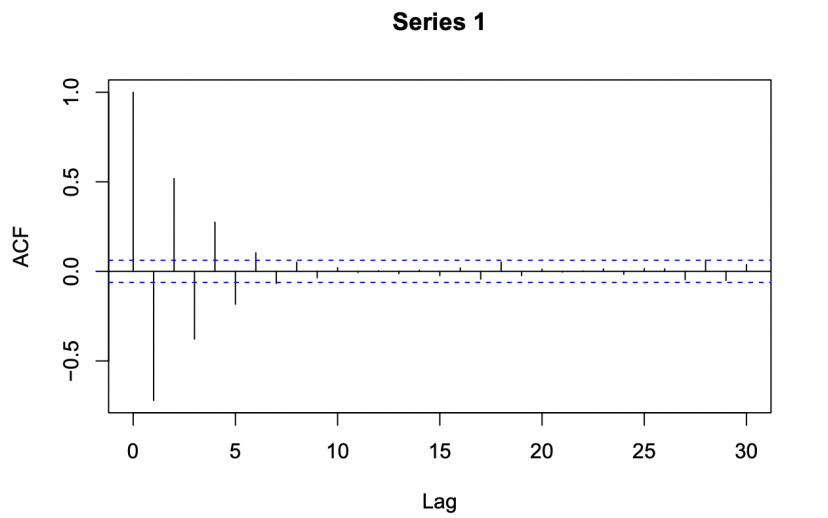
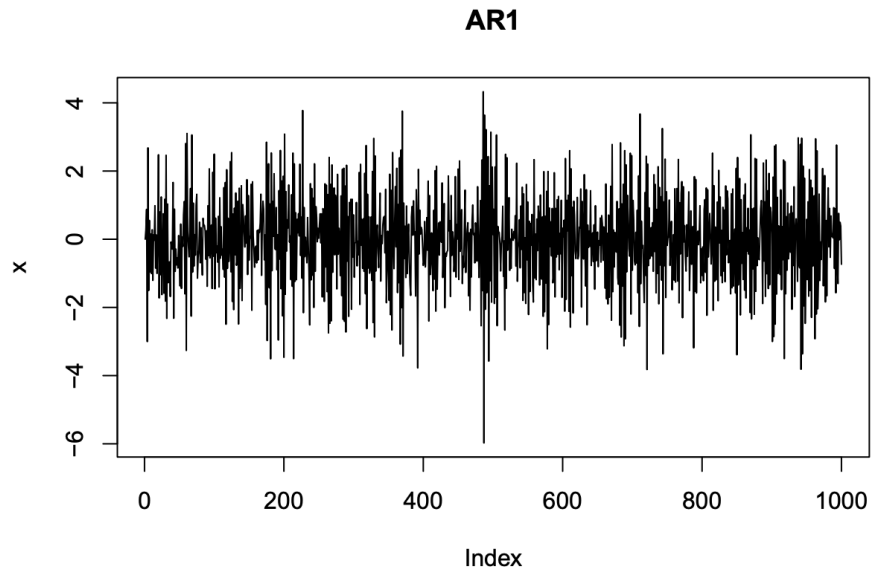
Mean and Std. dev:

Mean: -0.03455776

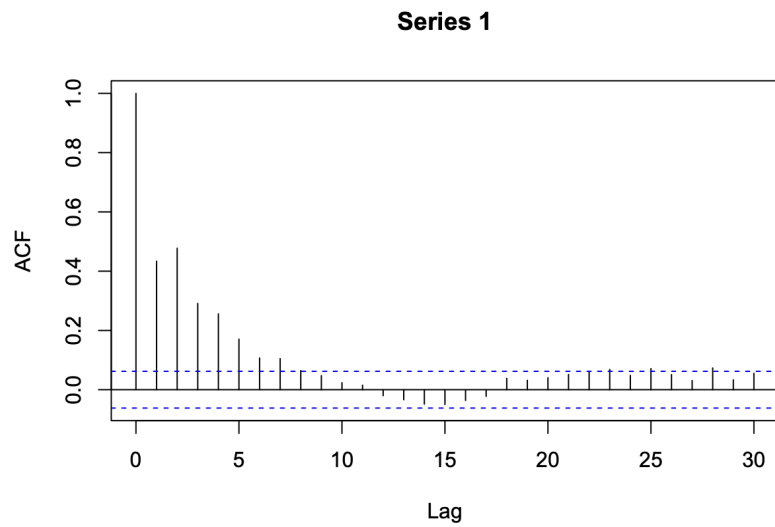
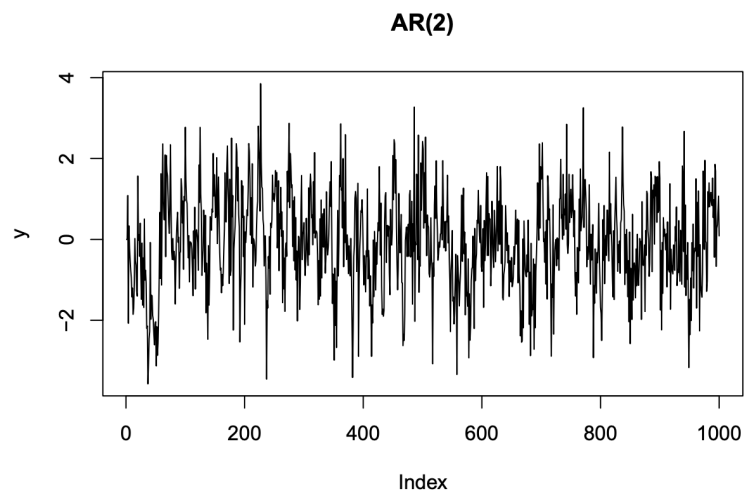
STD. deviation: 1.395705

Question 3

AR(1) process

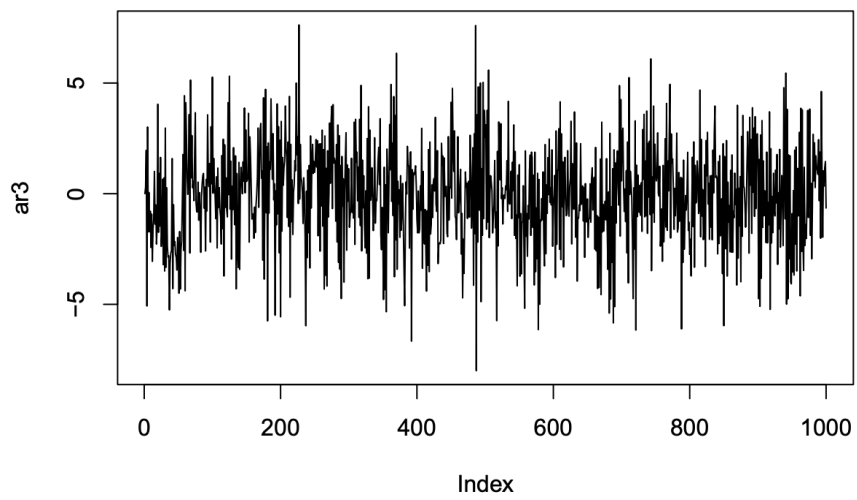


AR(2) process

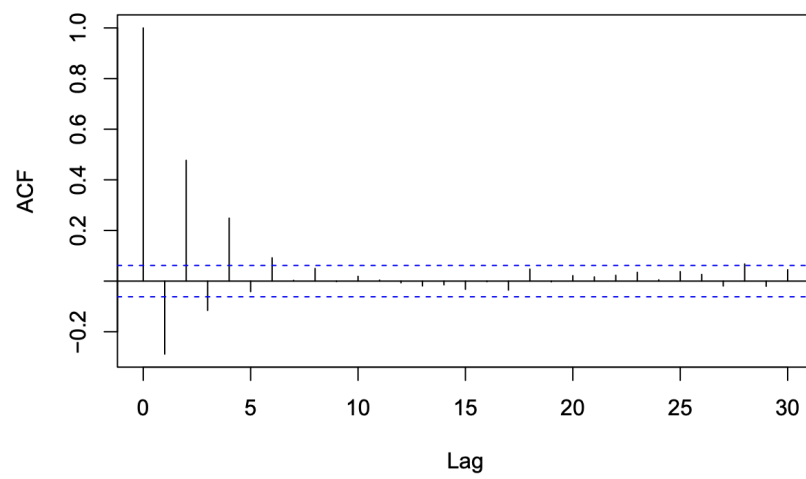


AR(1) + AR(2) Process

AR(1) + AR(2)




Series 1




Question 4

Monthly means:



Precipitation



Date	
1950-01-31	7.474498
1950-02-28	4.236547
1950-03-31	2.909981
1950-04-30	3.584058
1950-05-31	2.503505
...	...
2014-08-31	2.246686
2014-09-30	2.606470
2014-10-31	6.045129
2014-11-30	5.069574
2014-12-31	3.551625

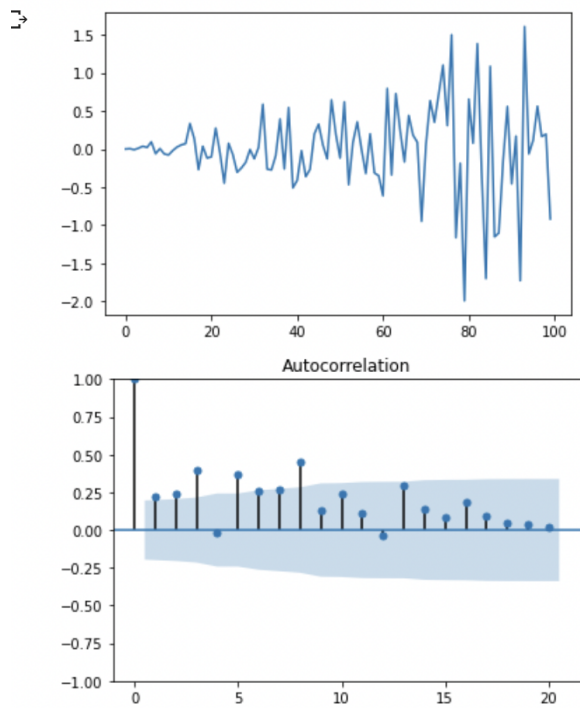
780 rows x 1 columns

Means:

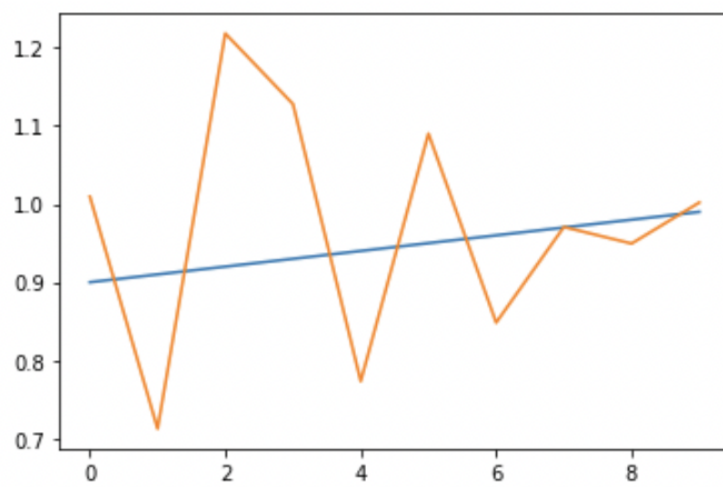
1. MSE without using the boxcox transform: 73.91905514465313
 2. MSE using the boxcox transform: 73.91905514465313
-

The **boxcox transform** hasn't made the MSE any better. Boxcox transforms do not always yield a better MSE. Probably a log transform could have worked better in our case.

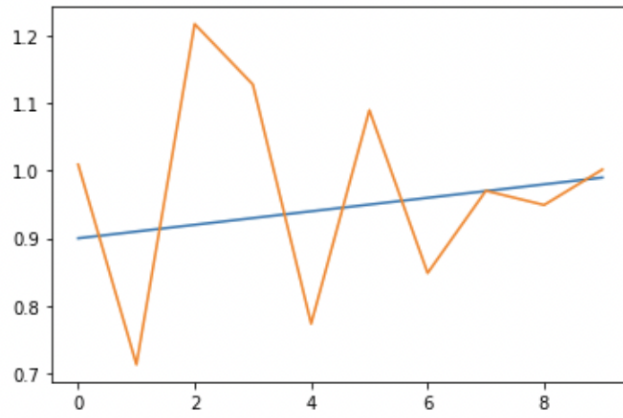
Question 5



Plot using ARCH model



Plot using GARCH



Question 6

Dr. Evan began to first speak about how the manner in which science is presented is important and how it has changed over the years. He highlighted how a punchline in the title of the paper or how if the information is presented in a better and more engaging way could attract more media and mass attention. He then spoke about how Affectiva harnesses computer vision in order to detect people's expressions/reactions while viewing advertisements. I thought this was particularly interesting as my other area of interest is people analytics and so this gave me more insights into how technology is leveraged in order to market engaging content to users. The next topic that was touched upon was network science. Two papers were spoken about under this hood. The node in this case was a train station and the link was the rail between them. A data driven approach was spoken of here to find out which one to bring back up first in the case of a disaster. The metric used was the state of critical functionality (where 1= the system is running correctly and 0= the system is disconnected). This was another interesting point for me, whereby this approach of making strategic data driven decisions to determine which node and link to bring up first based on how fast we could reach the state of critical functionality, could potentially save millions of dollars during the case of calamities. The second topic under the network science hood was the initiative that analyzed the United States Electric Grid as an Exchange Network. The paper focused on cutting down the greenhouse gas footprint of PCAs' (Power Control Authorities houses). The greenhouse gas footprint is according to the fossil fuel and renewable mix of energy that they have within their district. A network model was used as a transaction network to figure out what the actual carbon footprint is as opposed to what is being produced close to the PCA. Next Dr. Evans spoke about Climate risk and ESG analytics for fixed US income. The three pillars of what this was built on were- Physical Climate Risk, Social Impact and ESG (Environmental, Social and Governance Data) data and Carbon Transaction Risk. He showed us how different climate models perform different climate predictions for a given region. The idea was thus to combine the different climate models to give a single probability distribution. What was fascinating for me was the idea of integrating these climate models into the models that drive the insurance ecosystem. The focus then shifted to catastrophe modeling where the idea was to integrate GCM outputs into catastrophe models to help the reinsurance industry adapt to climate change. He goes on to explain as to why this strategy did not work- The models rely on historical data like what is a particular building made of, what will happen to it if it floods and so forth. The catastrophe modeling process in this case would deduce how much the annual damage costs would be for their portfolio. However, the reinsurer or insurer can change its prices each year and so it doesn't make sense for them to insure things that don't make sense business wise as their models are already good enough. It was extremely interesting to learn about the insurance gap, in particular. It was revealed that the insurance industry has managed to shift the burden of climate change over to the municipal level. Hence, Evan realized that asset managers are the target. The physical climate model also uses gcm to get the output probability distributions. The modification done was to change these outputs to monte carlo simulations. The climate conditioning method on the other hand uses a spatial model in order to identify the biases and test how these models perform in the real world and then carry those biases into future predictions. Another eye-opening aspect was the social impact of it all. It exposed how low income communities were at greater risk in the face of disaster. I enjoyed understanding how missing data was handled during the Zillow project code walkthrough. It showed me how to make sure of the available data more proactively. It was also helpful to see how Gaussian smoothing processes were used. After this lecture, I am more driven to understand all the various factors that make a strong relationship between climate risk and social vulnerability. To be able to model these factors in predictive models would serve as a huge strategic advantage as it would help increase accuracy while predicting social vulnerability indices. I'm enjoying these guest lectures and something that I'd like to learn from such forthcoming lectures would be to understand how time series is used in the Financial sector.

CIVE_Homework

Riddhi Narayan

2022-03-25

R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see <http://rmarkdown.rstudio.com>.

When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

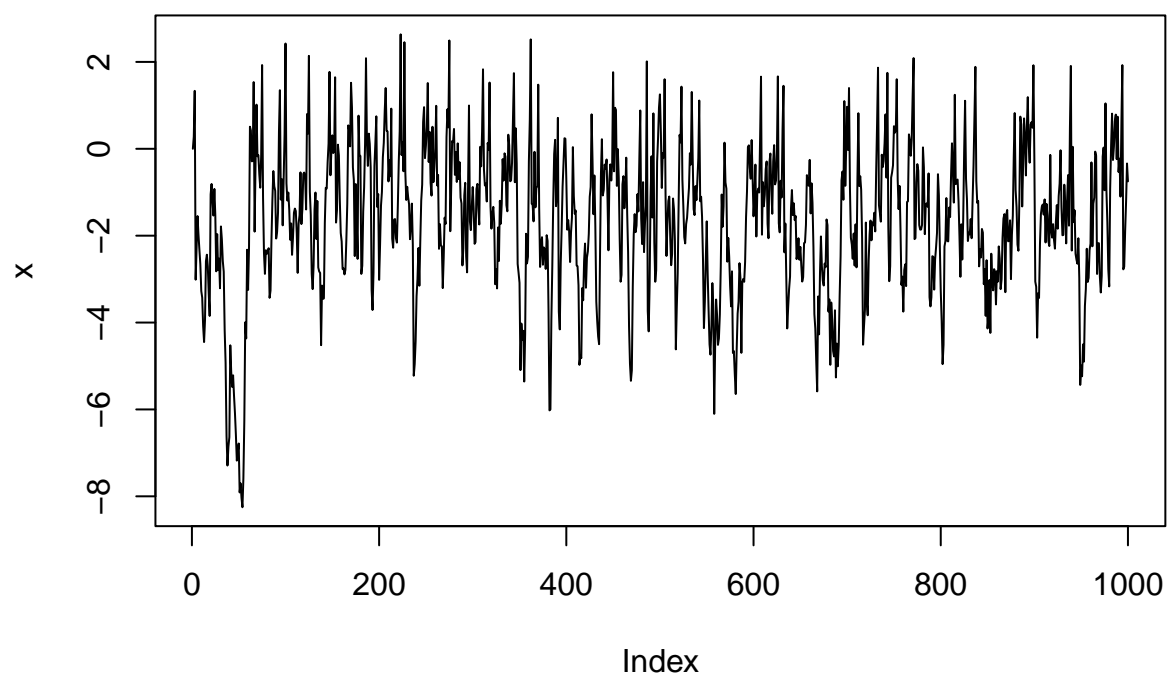
```
#Load the forecast package
library(forecast)
```

```
## Registered S3 method overwritten by 'quantmod':
##   method      from
##   as.zoo.data.frame zoo
```

```
# Set up variables
set.seed(1234)
n <- 1000
x <- matrix(0,1000,1)
w <- rnorm(n)

# loop to create x
for (t in 2:n){
  if(x[t-1] < 0.7){
    x[t] <- 0.9 * x[t-1] + w[t]
  }else{
    x[t] <- -0.5 * x[t-1] + w[t]
  }
}
plot(x,type='l', main='TAR model')
```

TAR model



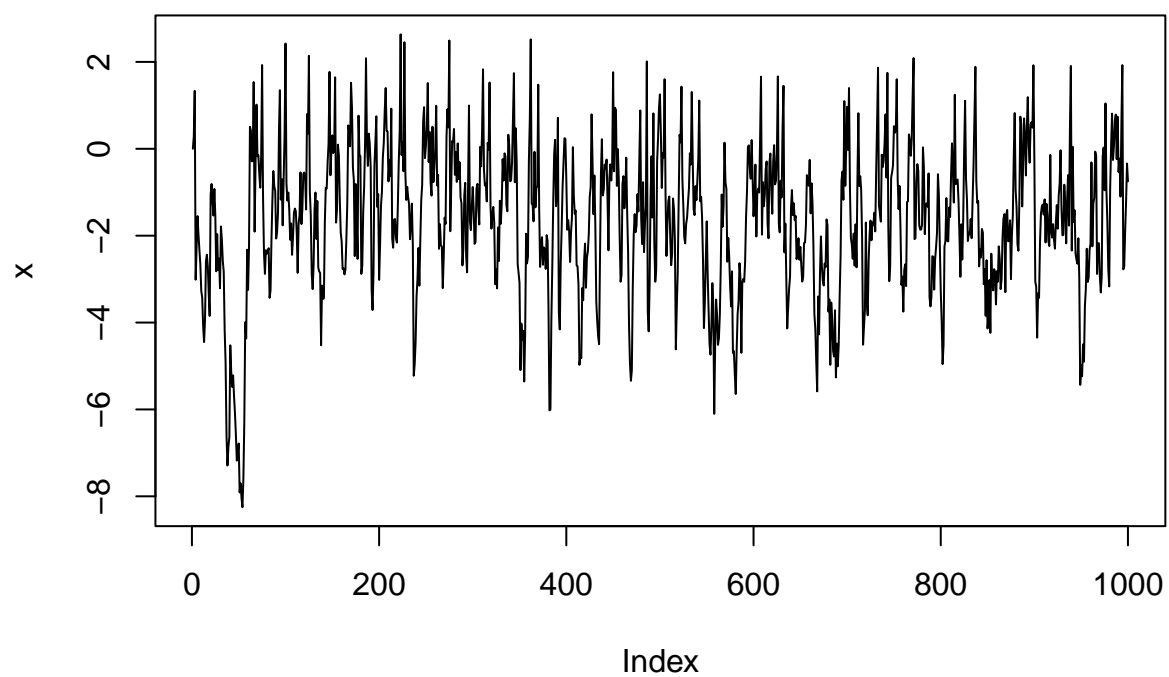
##Question 1

```
#Load the forecast package
library(forecast)

# Set up variables
set.seed(1234)
n <- 1000
x <- matrix(0,1000,1)
w <- rnorm(n)

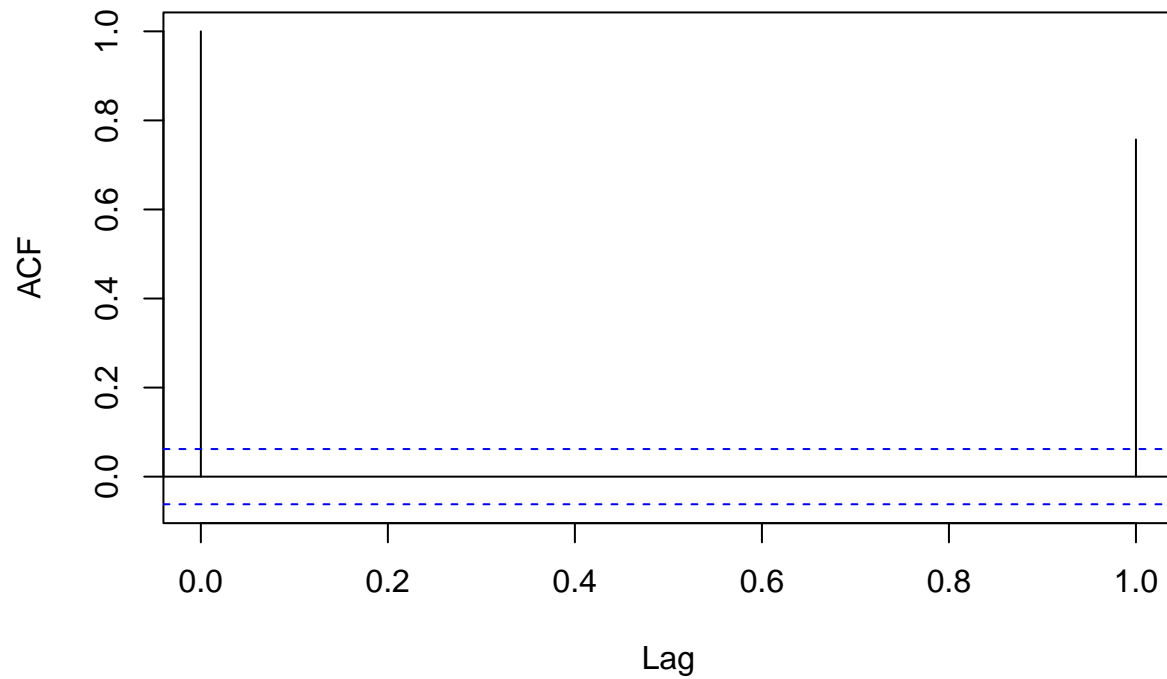
# loop to create x
for (t in 2:n){
  if(x[t-1] < 0.7){
    x[t] <- 0.9 * x[t-1] + w[t]
  }else{
    x[t] <- -0.5 * x[t-1] + w[t]
  }
}
plot(x,type='l', main='TAR model')
```

TAR model



```
#autocorrelation plot  
acf(x, lag.max = 1, plot=TRUE)
```

Series 1



```
#Mean and SD for Question1
```

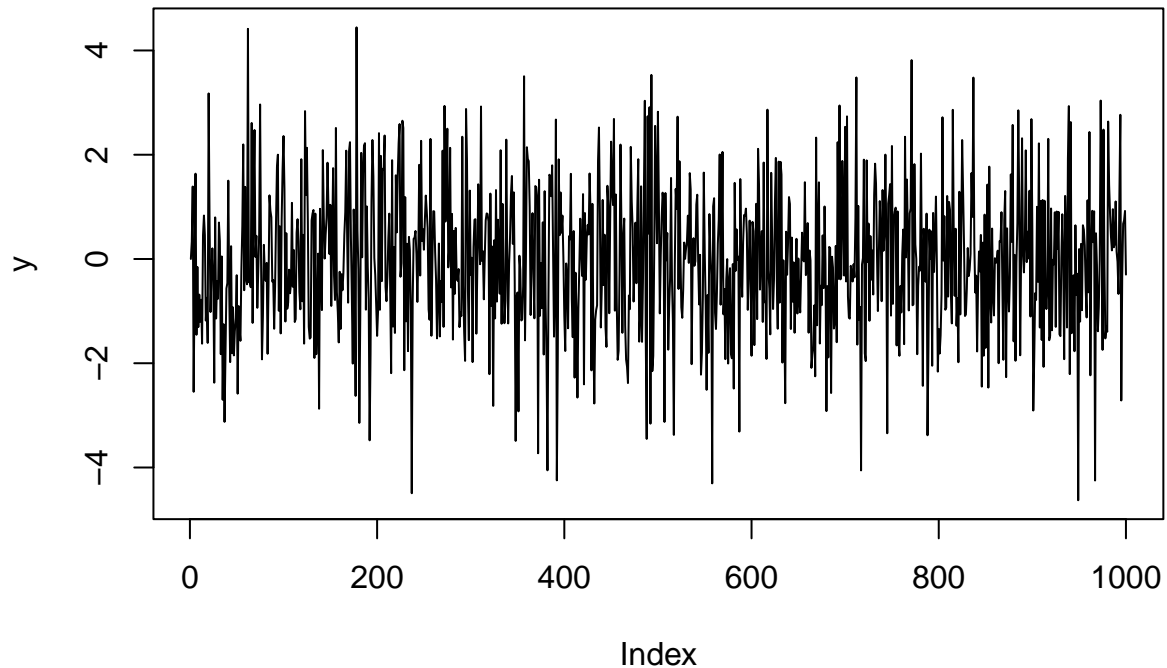
```
## [1] -1.642559
```

```
## [1] 1.710616
```

```
###Question 2
```

```
#Question 2  
#Generate TFN model  
# Set up variables  
set.seed(1234)  
n <- 1000  
x <- matrix(0,1000,1)  
y <- matrix(0,1000, 1)  
w <- rnorm(n)  
  
# loop to create x  
for (t in 2:n){  
  x[t] = -0.9 * x[t-1] + w[t]  
  y[t] <-0.2 *x[t] + 0.5 * x[t-1] + w[t]  
}  
plot(y,type='l', main='TFN model')
```

TFN model



```
#Mean and SD for Question2
```

```
mean(y)
```

```
## [1] -0.03455776
```

```
sd(y)
```

```
## [1] 1.395705
```

```
acf(y, max.lag=1, plot=TRUE)
```

```
## Warning in plot.window(...): "max.lag" is not a graphical parameter
```

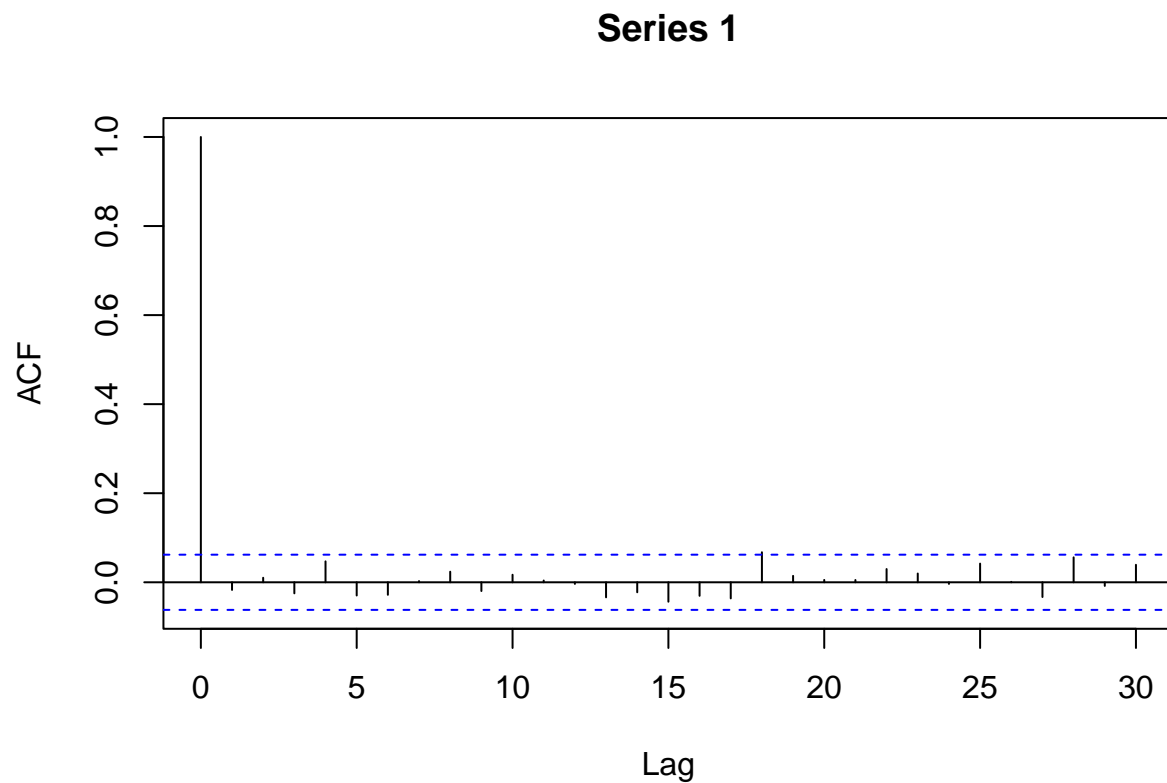
```
## Warning in plot.xy(xy, type, ...): "max.lag" is not a graphical parameter
```

```
## Warning in axis(side = side, at = at, labels = labels, ...): "max.lag" is not a  
## graphical parameter
```

```
## Warning in axis(side = side, at = at, labels = labels, ...): "max.lag" is not a  
## graphical parameter
```

```
## Warning in box(...): "max.lag" is not a graphical parameter
```

```
## Warning in title(...): "max.lag" is not a graphical parameter
```

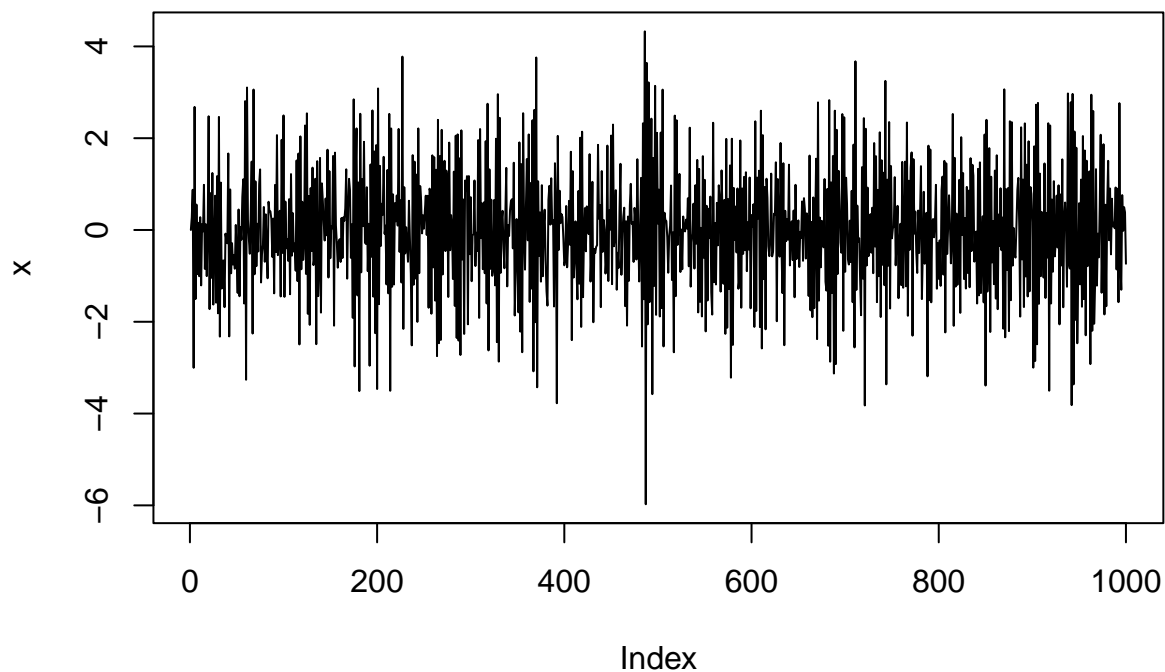


```
##Question 3
```

```
#Question 3
#Generate 3 (AR) models
# Set up variables
set.seed(1234)
n <- 1000
x <- matrix(0,1000,1)
y <- matrix(0,1000, 1)
ar3 <- matrix(0,1000,1)
w <- rnorm(n) #error term

# loop to create AR(1)
for (t in 2:n){
  x[t] = -0.75 * x[t-1] + w[t]
  #y[t] <- -0.2 * x[t] + 0.5 * x[t-1] + w[t]
}
plot(x,type='l', main='AR1')
```


AR1



```
acf(x, max.lag=1, plot=TRUE)
```

```
## Warning in plot.window(...): "max.lag" is not a graphical parameter
```

```
## Warning in plot.xy(xy, type, ...): "max.lag" is not a graphical parameter
```

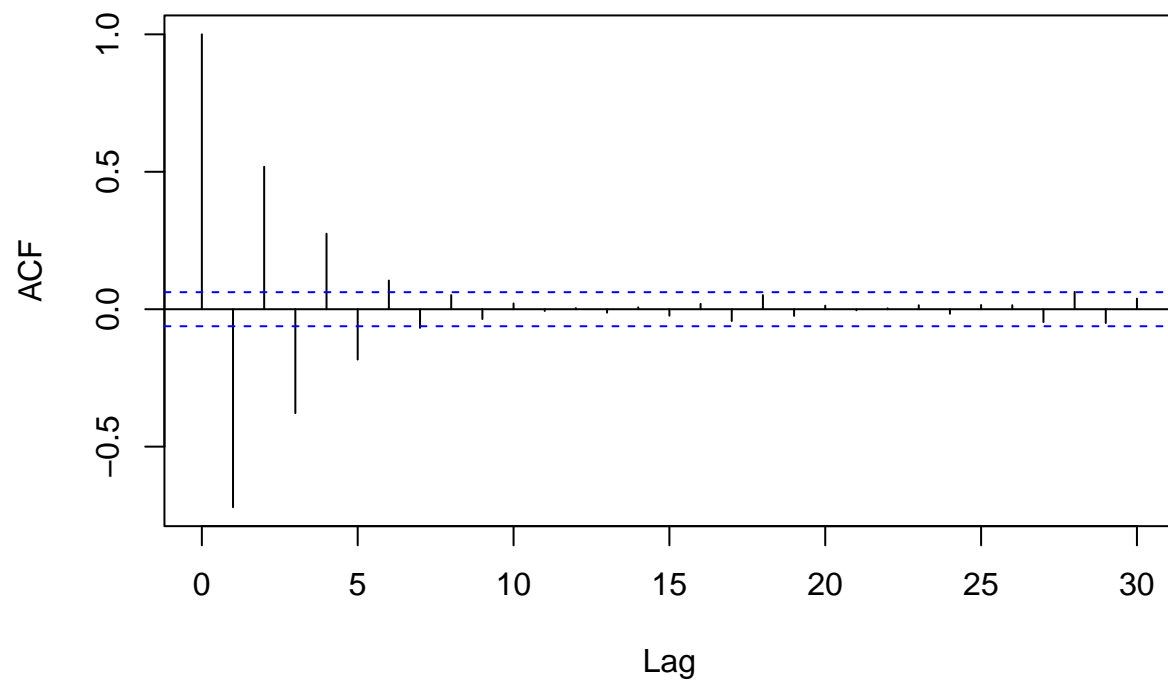
```
## Warning in axis(side = side, at = at, labels = labels, ...): "max.lag" is not a  
## graphical parameter
```

```
## Warning in axis(side = side, at = at, labels = labels, ...): "max.lag" is not a  
## graphical parameter
```

```
## Warning in box(...): "max.lag" is not a graphical parameter
```

```
## Warning in title(...): "max.lag" is not a graphical parameter
```

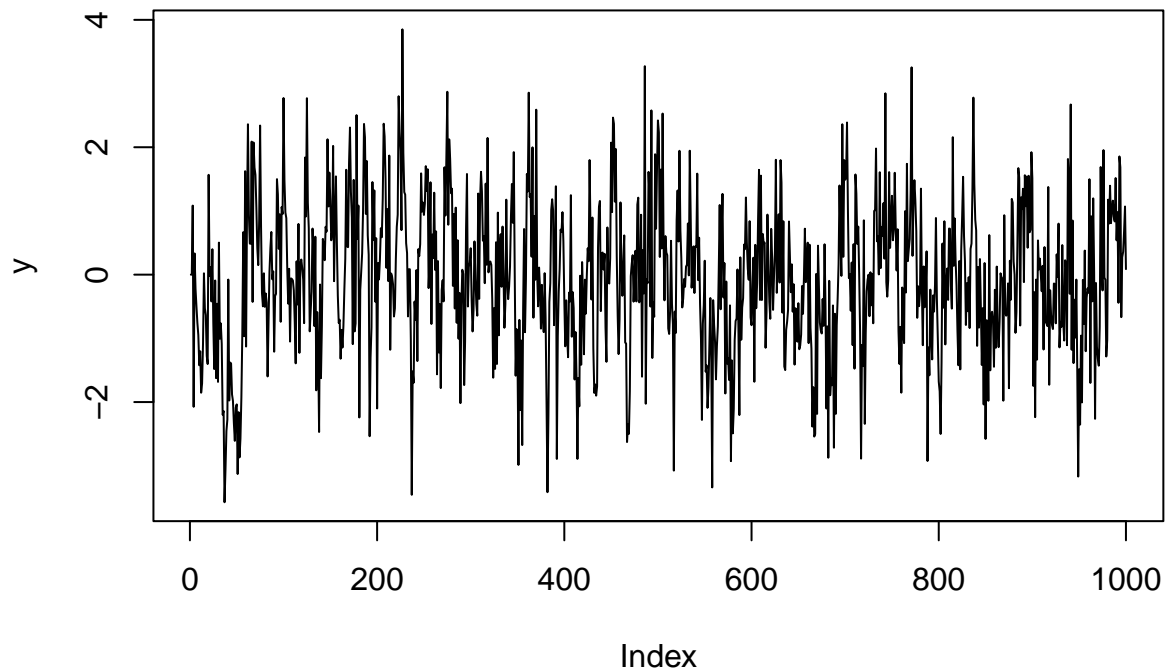
Series 1



```
#to create AR(2)
for (t in 3:n){
  y[t] = 0.25*y[t-1] + 0.39*y[t-2] + w[t]
}

plot(y,type='l', main='AR(2)')
```

AR(2)



```
acf(y, max.lag=1, plot=TRUE)
```

```
## Warning in plot.window(...): "max.lag" is not a graphical parameter
```

```
## Warning in plot.xy(xy, type, ...): "max.lag" is not a graphical parameter
```

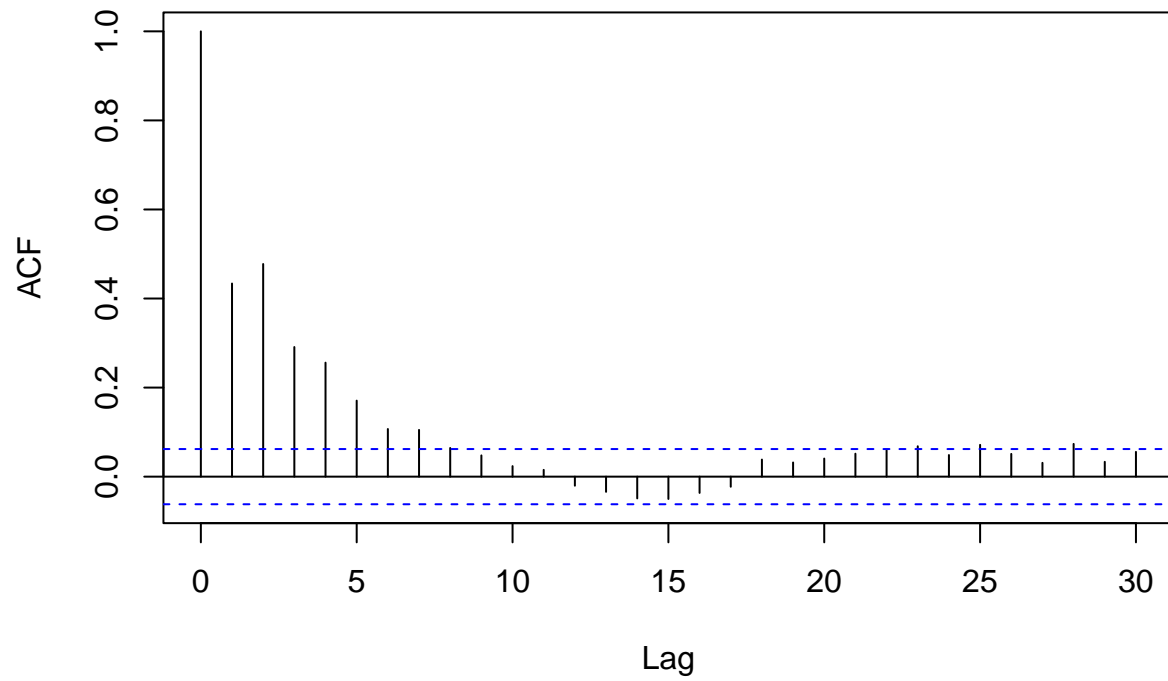
```
## Warning in axis(side = side, at = at, labels = labels, ...): "max.lag" is not a  
## graphical parameter
```

```
## Warning in axis(side = side, at = at, labels = labels, ...): "max.lag" is not a  
## graphical parameter
```

```
## Warning in box(...): "max.lag" is not a graphical parameter
```

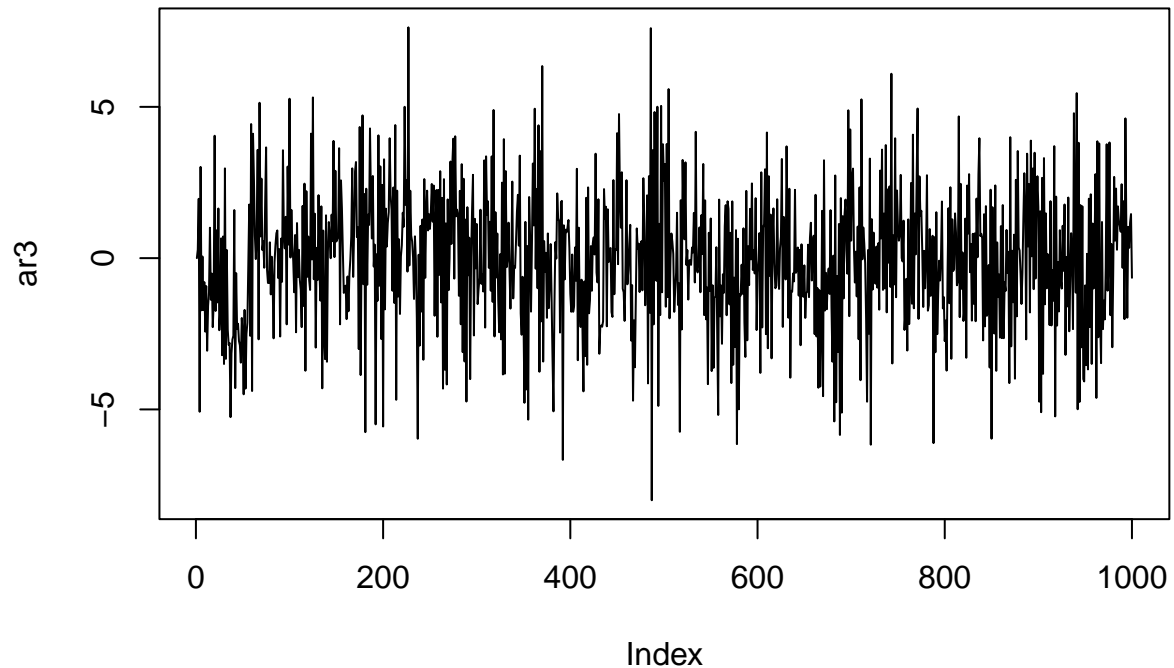
```
## Warning in title(...): "max.lag" is not a graphical parameter
```

Series 1



```
#plot AR(1) + AR(2)
for (t in 1:n){
  ar3[t] = x[t] + y[t]
}
plot(ar3, type='l', main='AR(1) + AR(2)')
```

AR(1) + AR(2)



```
acf(ar3, max.lag=1, plot=TRUE)
```

```
## Warning in plot.window(...): "max.lag" is not a graphical parameter
```

```
## Warning in plot.xy(xy, type, ...): "max.lag" is not a graphical parameter
```

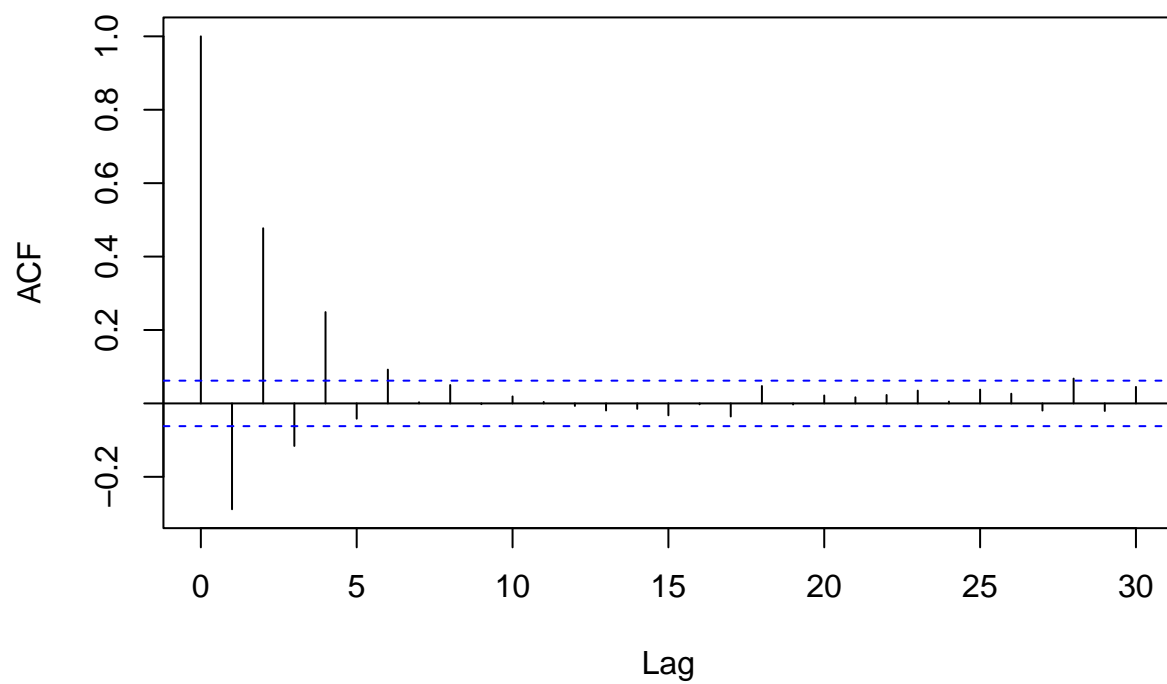
```
## Warning in axis(side = side, at = at, labels = labels, ...): "max.lag" is not a  
## graphical parameter
```

```
## Warning in axis(side = side, at = at, labels = labels, ...): "max.lag" is not a  
## graphical parameter
```

```
## Warning in box(...): "max.lag" is not a graphical parameter
```

```
## Warning in title(...): "max.lag" is not a graphical parameter
```

Series 1



#Looking at the 'AR(1) + AR(2) plot above and it's corresponding ACF plot, we can deduce that there is some new information added. The plot is more dense and the acf plot shows a lesser correlation with previous values.

▼ Question 4-

You are provided with daily annual rainfall in Boston for 65 years (1950-2014). Calculate monthly mean of the dataset. Use 50 years of data for training (using ARIMA process) and predict the Precipitation of Boston for the last 15 years. Report performance measure (MSE).

Try box-cox transformation where $\lambda=2$ and predict again. Retransform the dataset to initial form of the time series and find out the performance measure (MSE).

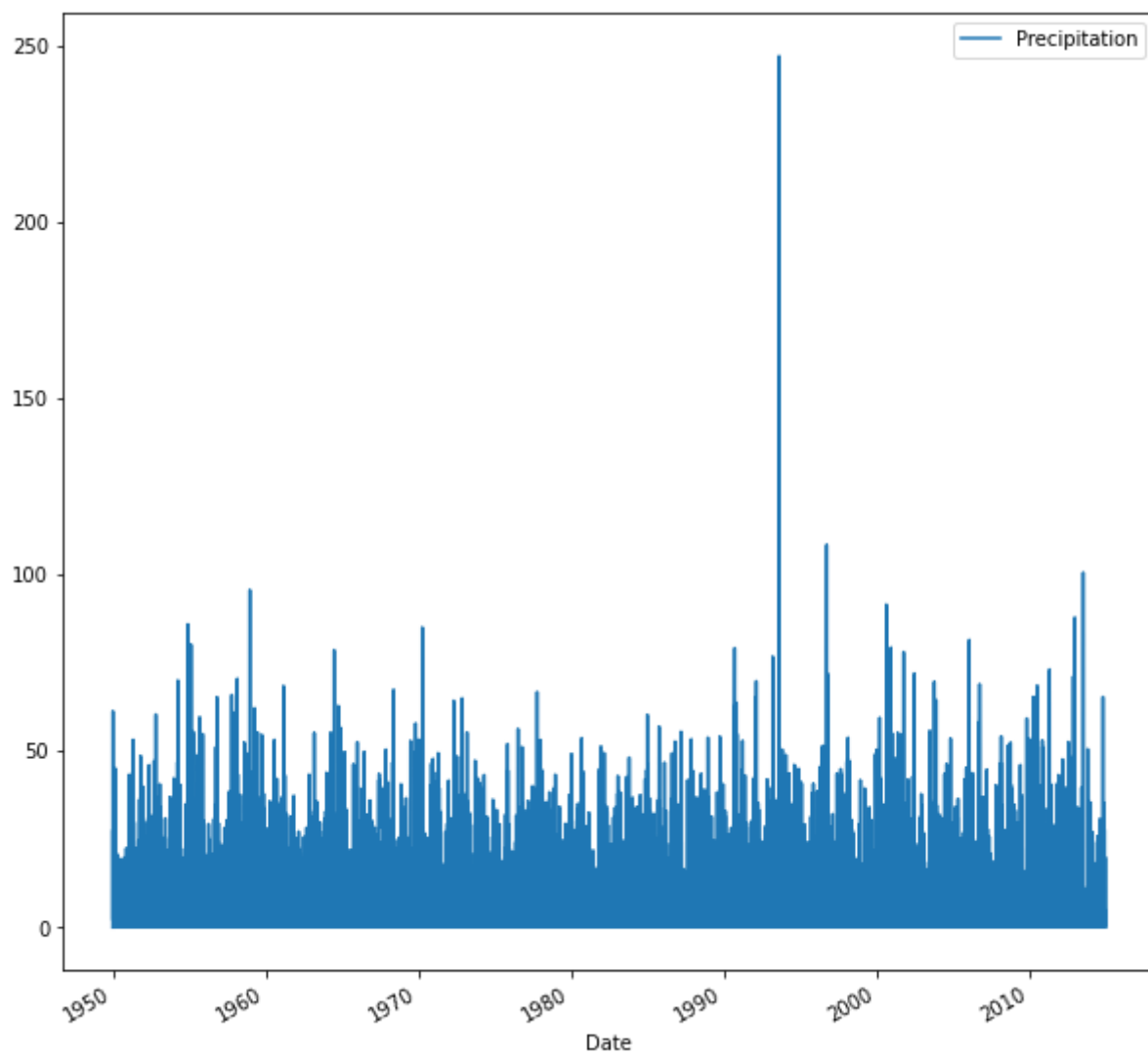
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
ts = pd.read_csv('/content/data_Precipitation.csv', index_col = 'Date', usecols=['Date', 'Precipitation'])
ts.head()
```

monthly mean

```
mm = pd.read_csv('/content/data_Precipitation.csv')
mm['Date'] = pd.to_datetime(mm['Date'])
mm.resample('M', on='Date').mean()
```

```
#plotting my data  
ts.plot(figsize=(10,10))  
plt.show()
```



```
#checking if data is stationary  
from statsmodels.tsa.stattools import adfuller  
results = adfuller(ts['Precipitation'])  
print(results[0])
```



```
print('P value is:', results[1])
```

```
-108.39772634860783
```

```
P value is: 0.0
```

▼ the above data is stationary

```
!pip install pmdarima
```

```
Requirement already satisfied: pmdarima in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: pandas>=0.19 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: statsmodels!=0.12.0,>=0.11 in /usr/local/lib/pyth
Requirement already satisfied: scikit-learn>=0.22 in /usr/local/lib/python3.7/di
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: urllib3 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: setuptools!=50.0.0,>=38.6.0 in /usr/local/lib/pyt
Requirement already satisfied: numpy>=1.19.3 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: Cython!=0.29.18,>=0.29 in /usr/local/lib/python3.
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: patsy>=0.5.2 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /usr/local/lib/python
```

```
from pmdarima import auto_arima
import warnings
warnings.filterwarnings('ignore')
```

```
Wise_fit = auto_arima(ts['Precipitation'], trace = True, suppress_warnings=True)
```

```
Performing stepwise search to minimize aic
ARIMA(2,0,2)(0,0,0)[0] intercept : AIC=167418.179, Time=20.06 sec
ARIMA(0,0,0)(0,0,0)[0] intercept : AIC=167799.476, Time=1.29 sec
ARIMA(1,0,0)(0,0,0)[0] intercept : AIC=167488.572, Time=1.50 sec
ARIMA(0,0,1)(0,0,0)[0] intercept : AIC=167455.838, Time=6.76 sec
ARIMA(0,0,0)(0,0,0)[0] : AIC=172002.343, Time=0.54 sec
ARIMA(1,0,2)(0,0,0)[0] intercept : AIC=167418.717, Time=24.50 sec
ARIMA(2,0,1)(0,0,0)[0] intercept : AIC=167416.599, Time=14.49 sec
ARIMA(1,0,1)(0,0,0)[0] intercept : AIC=167431.898, Time=13.21 sec
ARIMA(2,0,0)(0,0,0)[0] intercept : AIC=167415.898, Time=1.33 sec
ARIMA(3,0,0)(0,0,0)[0] intercept : AIC=167416.453, Time=3.06 sec
ARIMA(3,0,1)(0,0,0)[0] intercept : AIC=167418.447, Time=9.84 sec
ARIMA(2,0,0)(0,0,0)[0] : AIC=170268.896, Time=0.59 sec
```

```
Best model: ARIMA(2,0,0)(0,0,0)[0] intercept
Total fit time: 97.209 seconds
```

```
df2 = pd.read_csv('/content/data_Precipitation.csv')
df2['Date'] = pd.to_datetime(df2['Date'])

#Splitting into train and test
train = df2.loc[df2['Date'] <= '1999-12-31']
test = df2.loc[df2['Date'] >= '2000-01-01']

from statsmodels.tsa.arima.model import ARIMA
model_train = ARIMA(train['Precipitation'], order = (2,0,0))
model_train = model_train.fit()

model_train.summary()
```

```
test.head()
```

```
#testing part
import datetime
start = len(train)

end = len(train)+len(test)-1
test['forecast']= model_train.predict( start=start, end=end, dynamic=False)
#test[['Precipitation','forecast']].plot(figsize=(12,8))

test.head()
```

```
from sklearn.metrics import mean_squared_error

print('MSE without using the boxcox transform:',mean_squared_error(test1.Precipitatio

MSE without using the boxcox transform: 73.91905514465313
```

▼ using boxcox transformation

```
from scipy.stats import boxcox
df_pos = ts.loc[ts['Precipitation'] > 0]
df_pos['Precipitation'] = boxcox(df_pos['Precipitation'], lmbda=2)

#plotting my data
df_pos.plot(figsize=(10,10))
plt.show()
```

```
#checking if data is stationary
from statsmodels.tsa.stattools import adfuller
results = adfuller(df_pos['Precipitation'])
print(results[0])
print('P value is:', results[1])

-54.96325537226066
P value is: 0.0

from pmdarima import auto_arima
import warnings
warnings.filterwarnings('ignore')

good_fit = auto_arima(df_pos['Precipitation'], trace = True, suppress_warnings=True)

Performing stepwise search to minimize aic
ARIMA(2,0,2)(0,0,0)[0] intercept : AIC=313999.568, Time=8.44 sec
ARIMA(0,0,0)(0,0,0)[0] intercept : AIC=314019.859, Time=1.15 sec
ARIMA(1,0,0)(0,0,0)[0] intercept : AIC=313993.856, Time=1.34 sec
ARIMA(0,0,1)(0,0,0)[0] intercept : AIC=313994.401, Time=2.88 sec
ARIMA(0,0,0)(0,0,0)[0] : AIC=314560.038, Time=0.92 sec
ARIMA(2,0,0)(0,0,0)[0] intercept : AIC=313995.844, Time=2.04 sec
ARIMA(1,0,1)(0,0,0)[0] intercept : AIC=313995.853, Time=7.58 sec
ARIMA(2,0,1)(0,0,0)[0] intercept : AIC=313997.845, Time=12.64 sec
ARIMA(1,0,0)(0,0,0)[0] : AIC=314485.343, Time=0.32 sec

Best model: ARIMA(1,0,0)(0,0,0)[0] intercept
Total fit time: 37.351 seconds
```

```
df3 = pd.read_csv('/content/data_Precipitation.csv')
df3['Date'] = pd.to_datetime(df3['Date'])

#Splitting into train and test
train1 = df3.loc[df3['Date'] <= '1999-12-31']
test1 = df3.loc[df3['Date'] >= '2000-01-01']

from statsmodels.tsa.arima.model import ARIMA
model_boxcox = ARIMA(train1['Precipitation'], order = (1,0,0))
model_boxcox = model_boxcox.fit()
model_boxcox.summary()


#testing part
import datetime
start = len(train1)

end = len(train1)+len(test1)-1
test1['forecast'] = model_boxcox.predict( start=start, end=end, dynamic=False)

test1.head()
```

```
from sklearn.metrics import mean_squared_error

print('MSE using the boxcox transform:',mean_squared_error(test1.Precipitation, test1

MSE using the boxcox transform: 73.91905514465313
```

The boxcox transform hasn't made the MSE any better. Boxcox transforms do not always yield a better MSE. Probably a log transform could have worked better in our case.

▼ Question 5-

Simulate Model variance with ARCH and GARCH for Time Series Forecasting. Consider a time series of random noise where the mean is zero and the variance starts at 0.0 and steadily increases.

```
from random import gauss
from random import seed
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.graphics.tsaplots import plot_acf

#set seed
seed(4)
df = [gauss(0,i*0.01) for i in range (0, 100)]
df_sq = [ x**2 for x in df]

#plot the data
plt.plot(df)
plt.show()

#plot acf
```

```
plot_acf(np.asarray(df_sq))
plt.show()
```

significant positive correlatin till lag 15.

```
#train, test split
n_test = 10
train, test = df[:-n_test], df[-n_test:]
```

```
!pip install arch
```

```
Collecting arch
```

```
  Downloading arch-5.1.0-cp37-cp37m-manylinux_2_17_x86_64.manylinux2014_x86_64.w
  |██████████████████████████████████████████████████████████████████████████| 902 kB 5.3 MB/s
```

```
Requirement already satisfied: scipy>=1.3 in /usr/local/lib/python3.7/dist-packa
```

```
Requirement already satisfied: statsmodels>=0.11 in /usr/local/lib/python3.7/dis
```

```
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.7/dist-pack
```

```
Collecting property-cached>=1.6.4
```

```
  Downloading property_cached-1.6.4-py2.py3-none-any.whl (7.8 kB)
```

```
Requirement already satisfied: pandas>=1.0 in /usr/local/lib/python3.7/dist-pack
```

```
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-pac
```

```
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.
```

```
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-package
```

```
Requirement already satisfied: patsy>=0.5.2 in /usr/local/lib/python3.7/dist-pac
```

```
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /usr/local/lib/python
Installing collected packages: property-cached, arch
Successfully installed arch-5.1.0 property-cached-1.6.4
```

```
import arch
from arch import arch_model
model = arch_model(train, mean='Zero', vol='ARCH', p = 15)
```

```
modelfit = model.fit()
```

```
Iteration:      1,  Func. Count:      18,  Neg. LLF: 41.504522451939415
Iteration:      2,  Func. Count:      37,  Neg. LLF: 39.84160048057239
Iteration:      3,  Func. Count:      56,  Neg. LLF: 38.86152368289243
Iteration:      4,  Func. Count:      75,  Neg. LLF: 38.168837835452294
Iteration:      5,  Func. Count:      94,  Neg. LLF: 37.359456060716816
Iteration:      6,  Func. Count:     113,  Neg. LLF: 36.916153438658235
Iteration:      7,  Func. Count:     132,  Neg. LLF: 36.62927642773087
Iteration:      8,  Func. Count:     150,  Neg. LLF: 35.61568157968252
Iteration:      9,  Func. Count:     169,  Neg. LLF: 35.38236084253256
Iteration:     10,  Func. Count:     188,  Neg. LLF: 34.976965684955985
Iteration:     11,  Func. Count:     207,  Neg. LLF: 34.77947643918447
Iteration:     12,  Func. Count:     226,  Neg. LLF: 34.636329032030794
Iteration:     13,  Func. Count:     245,  Neg. LLF: 34.471421979520954
Iteration:     14,  Func. Count:     264,  Neg. LLF: 34.42026461625432
Iteration:     15,  Func. Count:     283,  Neg. LLF: 34.392186193914
Iteration:     16,  Func. Count:     302,  Neg. LLF: 34.36862004978023
Iteration:     17,  Func. Count:     320,  Neg. LLF: 34.364987911329905
Iteration:     18,  Func. Count:     338,  Neg. LLF: 34.36367196211154
Iteration:     19,  Func. Count:     356,  Neg. LLF: 34.36340962881595
Iteration:     20,  Func. Count:     374,  Neg. LLF: 34.363378087592224
Iteration:     21,  Func. Count:     392,  Neg. LLF: 34.363374904072806
Optimization terminated successfully.      (Exit mode 0)
      Current function value: 34.36337484867552
      Iterations: 21
      Function evaluations: 392
      Gradient evaluations: 21
```

```
#making predictions
pred = modelfit.forecast(horizon= n_test)
```

▼ Plot using ARCH model

```
#plotting variance and predicted variance

var = [i*0.01 for i in range(0,100)]
plt.plot(var[-n_test:])
plt.show()
```



```
plt.plot(pred.variance.values[-1, :])
plt.show()
```

```
#For the GARCH model
```

```
garch_model = arch_model(train, mean = 'Zero', vol = 'GARCH', p = 15)
garchfit = garch_model.fit()
```

```
Iteration:      1,  Func. Count:      19,  Neg. LLF: 43.295844959877854
Iteration:      2,  Func. Count:      44,  Neg. LLF: 43.243747831930406
Iteration:      3,  Func. Count:      64,  Neg. LLF: 38.99409977757669
Iteration:      4,  Func. Count:      84,  Neg. LLF: 38.219348236710395
Iteration:      5,  Func. Count:     104,  Neg. LLF: 37.77063814140541
Iteration:      6,  Func. Count:     124,  Neg. LLF: 36.85448822464735
Iteration:      7,  Func. Count:     144,  Neg. LLF: 36.28818440487556
Iteration:      8,  Func. Count:     164,  Neg. LLF: 36.018309596460604
Iteration:      9,  Func. Count:     184,  Neg. LLF: 35.83299193687124
Iteration:     10,  Func. Count:     204,  Neg. LLF: 35.72128260333963
Iteration:     11,  Func. Count:     224,  Neg. LLF: 35.24655924885105
Iteration:     12,  Func. Count:     244,  Neg. LLF: 35.0392646130409
Iteration:     13,  Func. Count:     264,  Neg. LLF: 34.84989174046595
Iteration:     14,  Func. Count:     284,  Neg. LLF: 34.713980500093484
Iteration:     15,  Func. Count:     303,  Neg. LLF: 34.442002549444915
Iteration:     16,  Func. Count:     322,  Neg. LLF: 34.37225246057246
Iteration:     17,  Func. Count:     342,  Neg. LLF: 34.36782095082328
Iteration:     18,  Func. Count:     361,  Neg. LLF: 34.36464767933175
Iteration:     19,  Func. Count:     380,  Neg. LLF: 34.36346035724202
Iteration:     20,  Func. Count:     399,  Neg. LLF: 34.36338081608357
Iteration:     21,  Func. Count:     418,  Neg. LLF: 34.36337487879584
```

```
Optimization terminated successfully.      (Exit mode 0)
```

```
Current function value: 34.36337490653557
```

```
Iterations: 21
```

```
Function evaluations: 418
```

```
Gradient evaluations: 21
```

```
pred1 = garchfit.forecast(horizon = n_test)
```

▼ Plot using GARCH model

```
#plotting variance and predicted variance
```

```
var1 = [i*0.01 for i in range(0,100)]
plt.plot(var1[-n_test:])
#plt.show()
plt.plot(pred1.variance.values[-1, :])
plt.show()
```

```
!jupyter nbconvert --to pdf /content/CIVE_Homework6_Q4&5.ipynb
```

```
/bin/bash: 5.ipynb: command not found
[NbConvertApp] WARNING | pattern '/content/CIVE_Homework6_Q4' matched no files
This application is used to convert notebook files (*.ipynb)
to various other formats.
```

```
WARNING: THE COMMANDLINE INTERFACE MAY CHANGE IN FUTURE RELEASES.
```

Options

=====

The options below are convenience aliases to configurable class-options, as listed in the "Equivalent to" description-line of the aliases.

To see all configurable class-options for some <cmd>, use:

```
<cmd> --help-all
```

--debug

```
set log level to logging.DEBUG (maximize logging output)
```

```
Equivalent to: [--Application.log_level=10]
```

--show-config

```
Show the application's configuration (human-readable format)
```

```
Equivalent to: [--Application.show_config=True]
```

--show-config-json

```
Show the application's configuration (json format)
```

```
Equivalent to: [--Application.show_config_json=True]
```

--generate-config

```
generate default config file
```

```
Equivalent to: [--JupyterApp.generate_config=True]
```

-y

```

    Answer yes to any questions instead of prompting.
    Equivalent to: [--JupyterApp.answer_yes=True]
--execute
    Execute the notebook prior to export.
    Equivalent to: [--ExecutePreprocessor.enabled=True]
--allow-errors
    Continue notebook execution even if one of the cells throws an error and inc
    Equivalent to: [--ExecutePreprocessor.allow_errors=True]
--stdin
    read a single notebook file from stdin. Write the resulting notebook with de
    Equivalent to: [--NbConvertApp.from_stdin=True]
--stdout
    Write notebook output to stdout instead of files.
    Equivalent to: [--NbConvertApp.writer_class=StdoutWriter]
--inplace
    Run nbconvert in place, overwriting the existing notebook (only
        relevant when converting to notebook format)
    Equivalent to: [--NbConvertApp.use_output_suffix=False --NbConvertApp.export
--clear-output
    Clear output of current file and save in place,
        overwriting the existing notebook.
    Equivalent to: [--NbConvertApp.use_output_suffix=False --NbConvertApp.export
--no-prompt
    Exclude input and output prompts from converted document.
    Equivalent to: [--TemplateExporter.exclude_input_prompt=True --TemplateExpor
--no-input
    Exclude input cells and output prompts from converted document.
        This mode is ideal for generating code-free reports.
    Equivalent to: [--TemplateExporter.exclude_output_prompt=True --TemplateExpc
--log-level=<Enum>

```