DSC 425

TIME SERIES ANALYSIS AND FORECASTING

ASSIGNMENT 7

PART 1

Problem 1

a)

```
# Problem 1
# a) Load the dataset, convert it into a time series with the proper time range, and plot the data vs the
# Load the dataset
data <- read.table("GDP_PerCap_US.txt", header = TRUE)

# Convert to time series
ts_data <- ts(data$GDP_Per_Cap, start = 1960, end = 2019)

# Plot the data vs the year
plot.ts(ts_data, main = "Per-capita GDP in the U.S. (1960-2019)", ylab = "GDP per capita", xlab = "Year")
```

Figure 1: Loading the dataset and converting it into time series

(Source: Developed in R Studio)

This figure represents loading the dataset in R studio and diverting it in the group of time series in a proper time range.

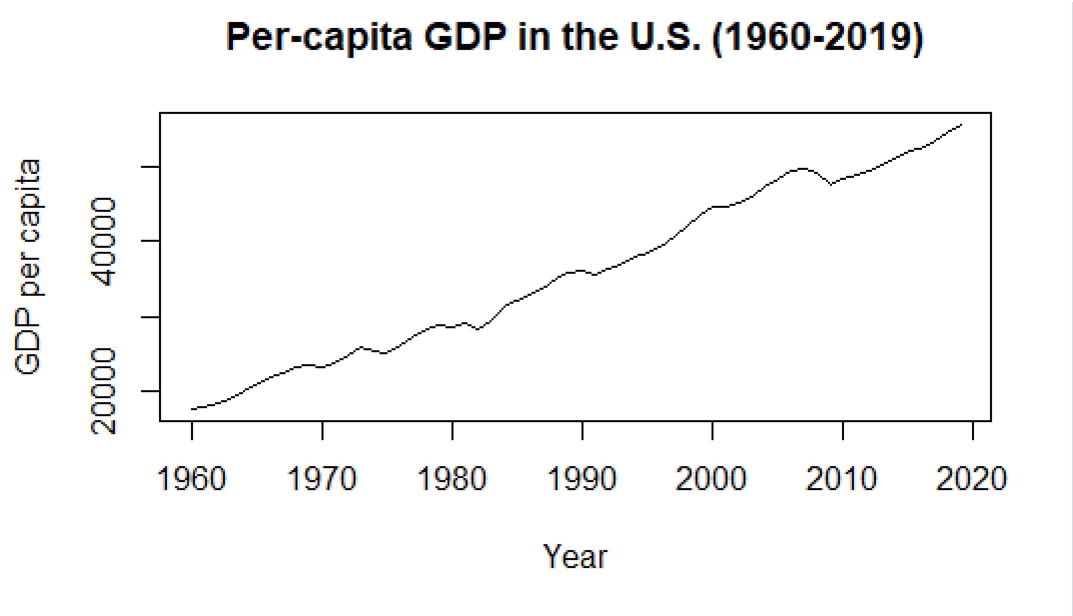


Figure 2: Plotting data vs year

(Source: Developed in R Studio)

The above figure shows plotting data vs year that is developed in R studio software.

b)

```
# b) Conduct a unit root test using Dickey-Fuller and KPSS
library(tseries)

# Dickey-Fuller test
adf.test(ts_data)

# KPSS test
kpss.test(ts_data)
```

Figure 3: Conduct a unit root test using Dicky-Fuller and KPSS

(Source: Developed in R Studio)

This image represents conducting a unit root test by using Dicky-Fuller and KPSS in R Studio software.

```
> # b) Conduct a unit root test using Dickey-Fuller and KPSS
> library(tseries)
> # Dickey-Fuller test
> adf.test(ts_data)

        Augmented Dickey-Fuller Test

data:  ts_data
Dickey-Fuller = -2.3317, Lag order = 3, p-value = 0.4407
alternative hypothesis: stationary

> # KPSS test
> kpss.test(ts_data)

        KPSS Test for Level Stationarity

data:  ts_data
KPSS Level = 1.601, Truncation lag parameter = 3, p-value = 0.01
```

Figure 4: Dicky-Fuller and KPSS Test

(Source: Developed in R Studio)

This figure highlights the result of Dicky-Fuller is -2.3317, lag order is 3, the p-value is 0.01, and KPSS level = 1.601.

c)

```
# c) Run a simple OLS linear model and analyze the residuals
# Create a time variable
time <- 1960:2019

# Fit the OLS linear model
model <- lm(ts_data ~ time)

# Analyze residuals for non-stationarity
library(lmtest)

# ACF/PACF
acf(model$residuals)
pacf(model$residuals)

# Unit root test
adf.test(model$residuals)

# KPSS test
kpss.test(model$residuals)
```

Figure 5: Run a simple OLS linear model and analyze the residuals

(Source: Developed in R Studio)

This image highlights the process of analyzing the residuals after running an easy OLS linear model on the basis of the unit root test and KPSS test.

```
> # c) Run a simple OLS linear model and analyze the residuals
> # Create a time variable
> time <- 1960:2019
> # Fit the OLS linear model
> model <- lm(ts_data ~ time)
> # Analyze residuals for non-stationarity
> library(lmtest)
> # ACF/PACF
> acf(model$residuals)
> pacf(model$residuals)
> # Unit root test
> adf.test(model$residuals)

        Augmented Dickey-Fuller Test

data:  model$residuals
Dickey-Fuller = -2.3317, Lag order = 3, p-value = 0.4407
alternative hypothesis: stationary

> # KPSS test
> kpss.test(model$residuals)

        KPSS Test for Level Stationarity

data:  model$residuals
KPSS Level = 0.14504, Truncation lag parameter = 3, p-value = 0.1
```

Figure 6: Result of ACF/PACF and unit root tests.

(Source: Developed in R Studio)

The above figure deploys the result of KPSS level is 0.14504, Dickey Fuller is -2.3317, and truncation lag parameter is 3.

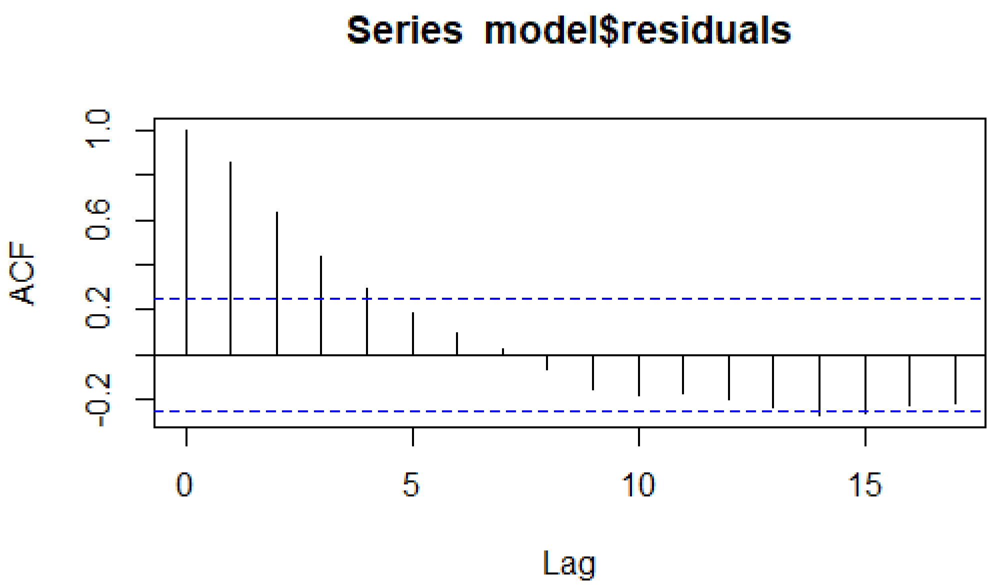


Figure 7: Plotting of series model and residuals

(Source: Developed in R Studio)

The plotting of the series model and residuals can be developed on the above image by implementing R studio software.

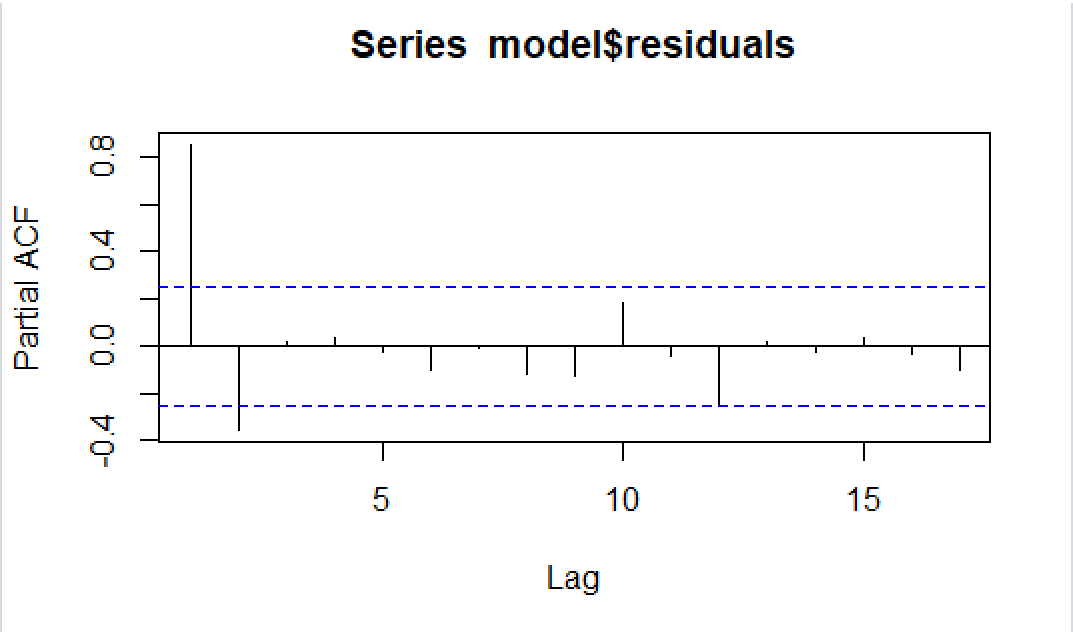


Figure 7: Series model and residuals graphic on partial ACF

(Source: Developed in R Studio)

This image highlights the series model and residuals graph on the partial ACF using R studio software based on partial ACF.

d)

```
# d) Analyze ACF/PACF/EACF of both the series and the regression residuals
# ACF/PACF of the series
acf(ts_data)
pacf(ts_data)

# ACF/PACF of the residuals
acf(model$residuals)
pacf(model$residuals)
```

Figure 8: Analyzing ACF/PACF/EACF of series and regression residuals

(Source: Developed in R Studio)

This figure represents an analysis of the ACF, PACF, EACF, and regression residuals for the series.

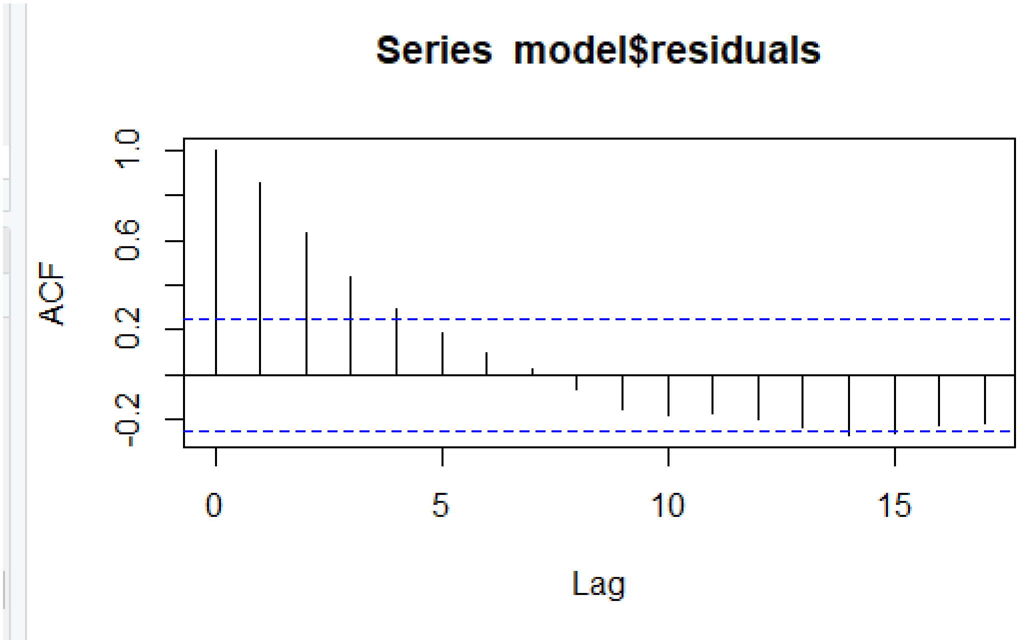


Figure 9: Plotting of series and regression residuals

(Source: Developed in R Studio)

Series plotting and regression residuals have been plotted on the above image on the basis of ACF.

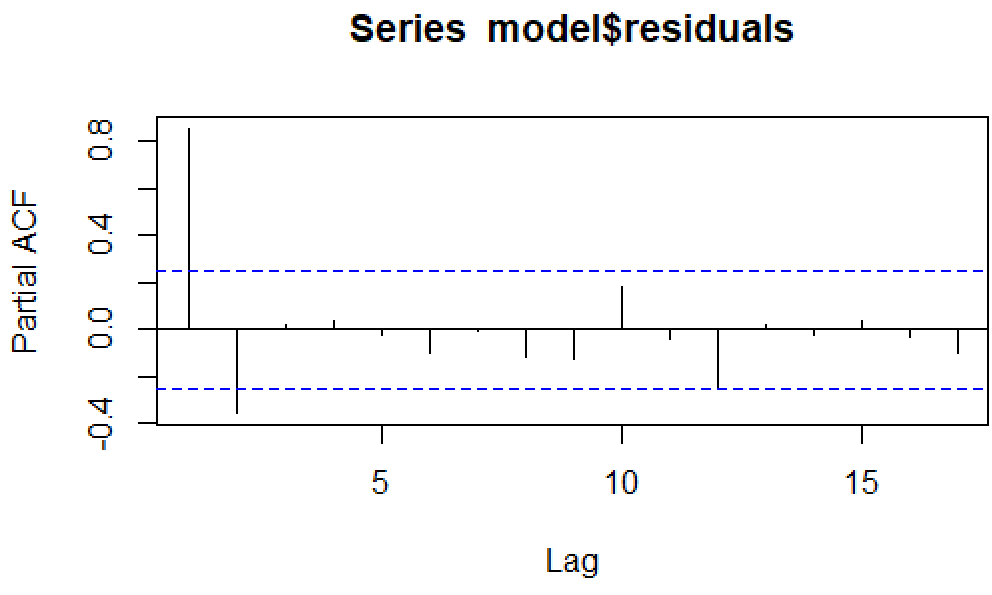


Figure 9: Series and regression residuals on partial ACF

(Source: Developed in R Studio)

This image highlights series and regression residuals on a partial ACF implemented in R Studio software.

e)

```
# e) Fit an appropriate ARIMA model, test coefficients for significance, and analyze resid
library(forecast)

# Fit ARIMA model
arima_model <- auto.arima(ts_data)

# Test coefficients for significance
coeftest(arima_model)

# Analyze residuals for autocorrelation
Box.test(arima_model$residuals)
```

Figure 9: Fitting an appropriate ARIMA model

(Source: Developed in R Studio)

This image expresses a suitable ARIMA model that is fitted in R Studio software for analyzing residuals for autocorrelation.

```
> # e) Fit an appropriate ARIMA model, test coefficients for significance, and analyze residuals
> library(forecast)
> # Fit ARIMA model
> arima_model <- auto.arima(ts_data)
> # Test coefficients for significance
> coeftest(arima_model)

z test of coefficients:

       Estimate Std. Error z value  Pr(>|z|)
ma1     0.33925    0.12147  2.7930  0.005223 **
drift 643.50667  102.23613  6.2943 3.088e-10 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

> # Analyze residuals for autocorrelation
> Box.test(arima_model$residuals)

        Box-Pierce test

data:  arima_model$residuals
X-squared = 0.00015985, df = 1, p-value = 0.9899
```

Figure 10: Result of Fitting an Appropriate ARIMA Model

(Source: Developed in R Studio)

This image shows X -the squared value is 0.00015985, df is 1, and the p-value is 0.9899 on the impact of the ARIMA model.

f)

```
# f) Compare the model built with auto.arima and run a backtest
# Compare models
arima_model
auto_arima_model <- forecast::auto.arima(ts_data)
auto_arima_model
# Backtest comparison
accuracy(arima_model)
accuracy(auto_arima_model)
```

Figure 11: Comparing the model built with auto. arima and running a backtest

(Source: Developed in R Studio)

This image shows the comparison of the model created using auto. Arima and backtest in the prediction aspects.

```
> # f) Compare the model built with auto.arima and run a backtest
> # Compare models
> arima_model
Series: ts_data
ARIMA(0,1,1) with drift

Coefficients:
         ma1     drift
      0.3392  643.5067
s.e.  0.1215  102.2361

sigma^2 = 358893:  log likelihood = -460.09
AIC=926.18   AICc=926.61   BIC=932.41
> auto_arima_model <- forecast::auto.arima(ts_data)
> auto_arima_model
Series: ts_data
ARIMA(0,1,1) with drift

Coefficients:
         ma1     drift
      0.3392  643.5067
s.e.  0.1215  102.2361

sigma^2 = 358893:  log likelihood = -460.09
AIC=926.18   AICc=926.61   BIC=932.41
> # Backtest comparison
> accuracy(arima_model)
                   ME     RMSE      MAE        MPE     MAPE      MASE       ACF1
Training set 3.046825 583.9076 462.0586 -0.0366596 1.402162 0.5692753 0.00163225
> accuracy(auto_arima_model)
                   ME     RMSE      MAE        MPE     MAPE      MASE       ACF1
Training set 3.046825 583.9076 462.0586 -0.0366596 1.402162 0.5692753 0.00163225
```

Figure 12: Result of comparing the model built with auto. arima and running a backtest

(Source: Developed in R Studio)

This image precedes comparing the model ARIMA (0,0,1) with drift coefficients and AIC is 926.18, BIC is 932.41, and AICc is 926.61.

g)

```
# g) Run a 10-year forecast for the per-capita GDP
# 10-year forecast
forecast_arima <- forecast(arima_model, h = 10)
forecast_auto_arima <- forecast(auto_arima_model, h = 10)

# Plot forecasts
plot(forecast_arima, main = "ARIMA Forecast for Per-capita GDP (Next 10 years)")
plot(forecast_auto_arima, main = "Auto ARIMA Forecast for Per-capita GDP (Next 10 years)")
```

Figure 13: Running a 10-year forecast for the per-capita GDP

(Source: Developed in R Studio)

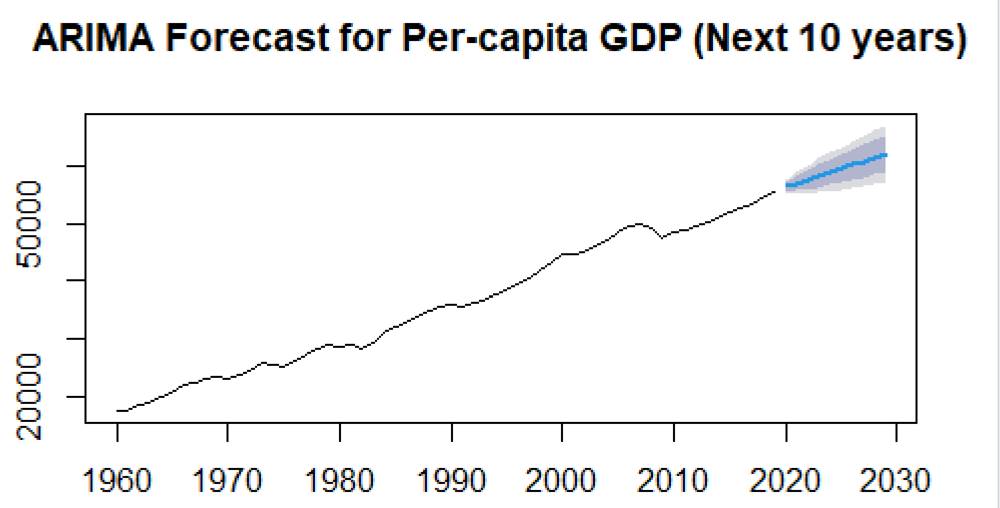


Figure 14: Plotting of ARIMA forecast for per-capita GDP in the next 10 years

(Source: Developed in R Studio)

This figure enumerates the next ten years' per-capita GDP estimate using ARIMA developed here.

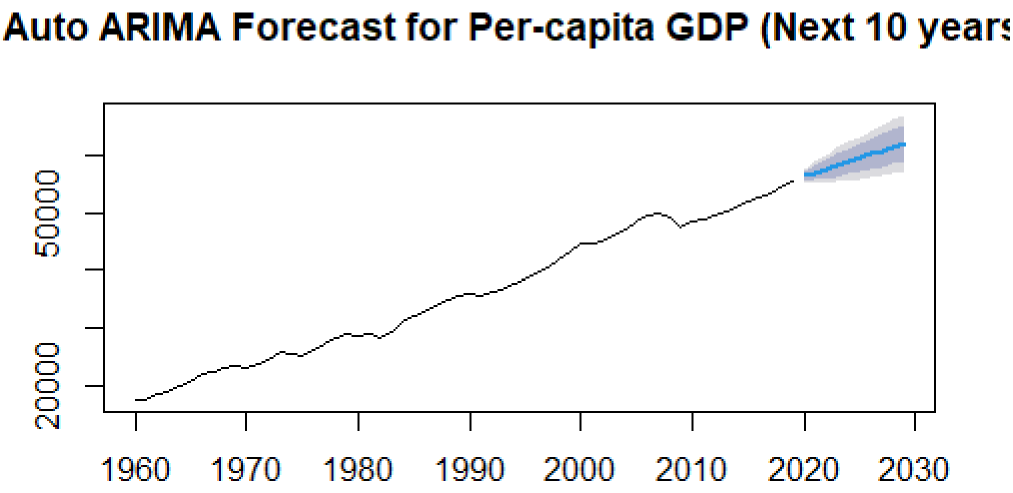


Figure 15: Plotting of Auto ARIMA forecast for per-capita GDP in the next 10 years

(Source: Developed in R Studio)

This figure highlights auto ARIMA forecasting for per-capita GDP for the next ten years in R Studio.

Problem 2

a)

```
# Problem 2

# a)
# Load the dataset
data <- read.csv("EnergyConsumption.csv")

# Convert the date column to Date type
data$date <- as.Date(data$date, format = "%m/%d/%Y")

# Create a time series object
ts_data <- ts(data$Energy, start = c(1973, 1), frequency = 12)

# Plot the Energy variable
plot.ts(ts_data, main = "Energy Consumption Over Time", ylab = "Energy (MW)", xlab = "Year")
```

Figure 16: Loading the dataset and converting the data column to data type

(Source: Developed in R Studio)

This image develops the dataset loaded after the data column is changed to a data type.

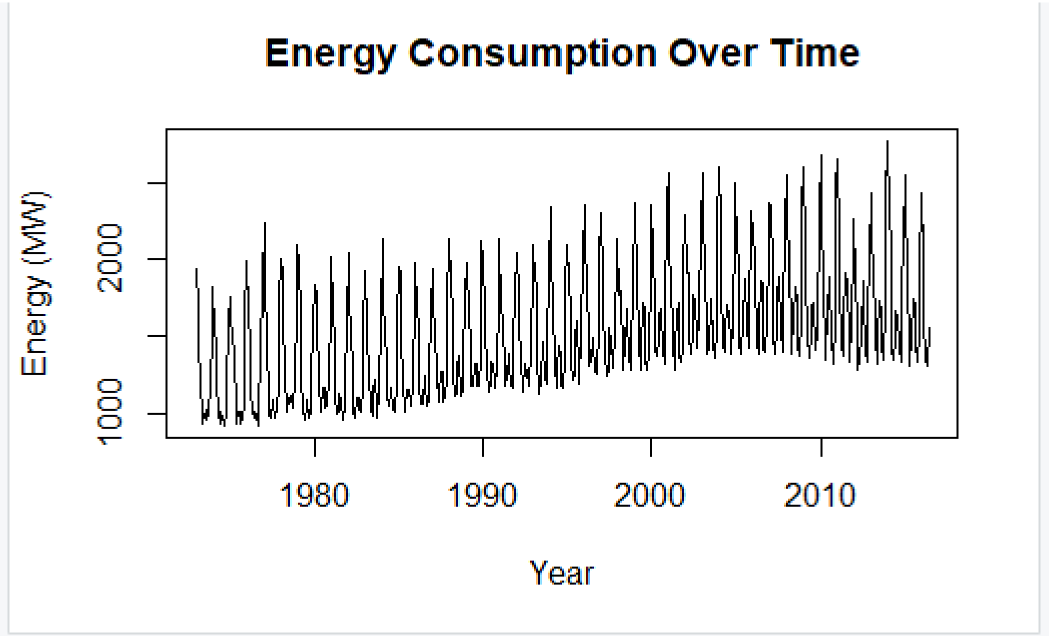


Figure 17: Plotting energy consumption over time

(Source: Developed in R Studio)

This figure objectifies plotting the amount of energy used with time developed in R studio.

b)

```
# b)
library(forecast)

# ACF plot of Energy
acf(ts_data, lag.max = 20)
```

Figure 18: Library importing for ACF plot of energy

(Source: Developed in R Studio)

Library importing for ACF plot of energy is done on the above image developed in R studio.

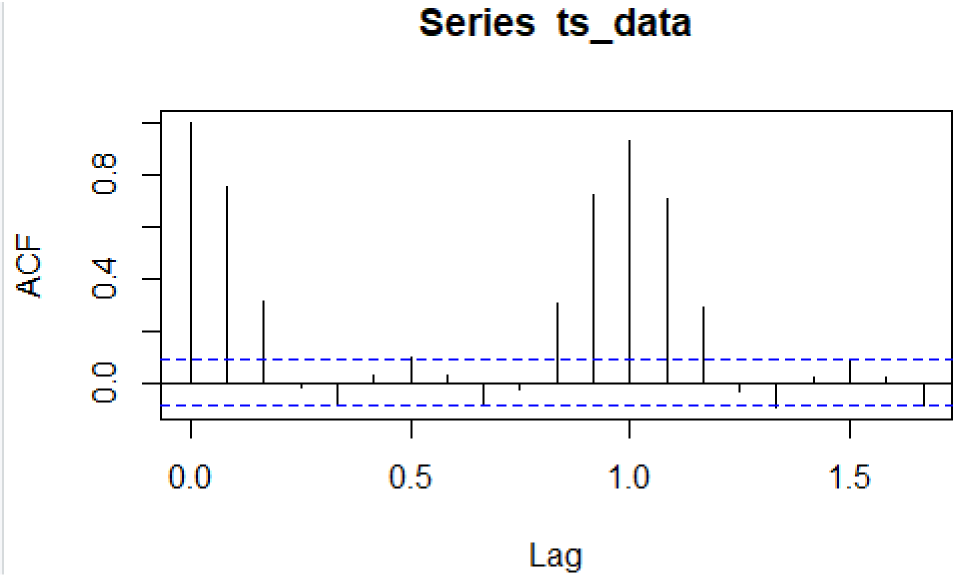


Figure 19: Plotting series vs data

(Source: Developed in R Studio)

This figure represents plotting series vs data that are developed in R Studio for ACF.

c)

```
# c)
# First difference of Energy
diff_ts_data <- diff(ts_data)

# ACF plot of the first difference of Energy
acf(diff_ts_data, lag.max = 20)
```

Figure 20: ACF plotting of first difference of energy

(Source: Developed in R Studio)

The image derives the ACF plotting of the first difference of energy developed in R studio.

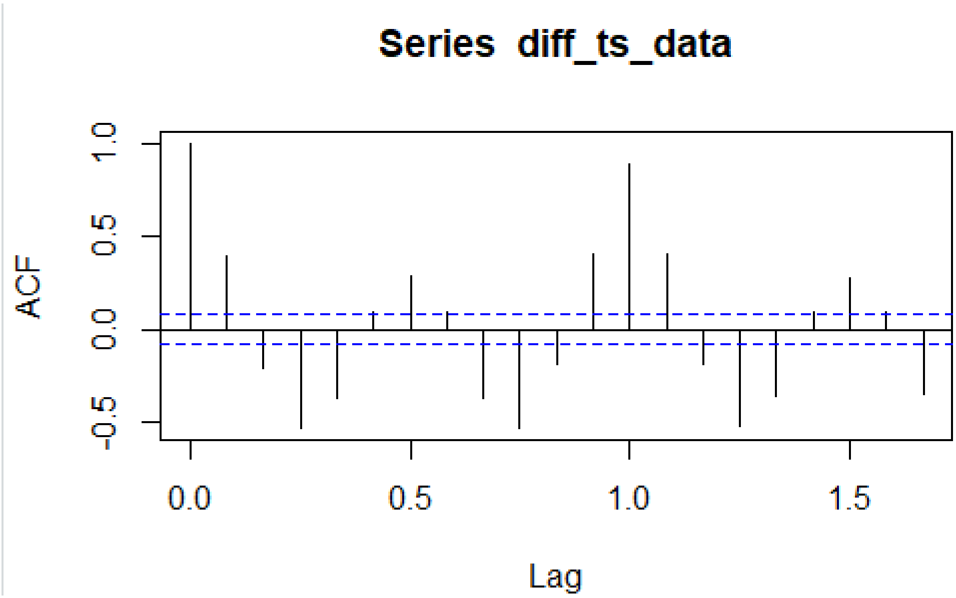


Figure 21: Plotting ACF function (20 lags) of the first difference of Energy

(Source: Developed in R Studio)

Plotting the ACF function of 20 lags of the first comparison of energy is conducted in R studio software.

d)

```
# d)
library(urca)

# Unit-root test
summary(ur.df(ts_data))
```

Figure 22: Library imported for unit root test

(Source: Developed in R Studio)

The library (urca) has been imported for the development of the unit root test shown in the above image.

```
R 4.2.3 · ~/
> library(urca)
> # Unit-root test
> summary(ur.df(ts_data))

###############################################
# Augmented Dickey-Fuller Test Unit Root Test #
###############################################

Test regression none


Call:
lm(formula = z.diff ~ z.lag.1 - 1 + z.diff.lag)

Residuals:
    Min      1Q  Median      3Q     Max
-706.22 -116.90   51.64  193.56  802.44

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
z.lag.1    -0.021228   0.007001  -3.032  0.00255 **
z.diff.lag  0.405489   0.040139  10.102  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 251.1 on 518 degrees of freedom
Multiple R-squared:  0.1711,	Adjusted R-squared:  0.1679
F-statistic: 53.45 on 2 and 518 DF,  p-value: < 2.2e-16


Value of test-statistic is: -3.0323

Critical values for test statistics:
      1pct  5pct 10pct
tau1 -2.58 -1.95 -1.62
```

Figure 23: Critical values for test statistics

(Source: Developed in R Studio)

The above figure shows the value of test statistics is -3.0323 and the critical values of test statistics for tau1 in 1pct, 5pct, and 10pct are -2.58, -1.95, and -1.62 respectively.

e)

```
# e)
# Manually build ARIMA model
model1 <- Arima(ts_data, order = c(1, 0, 0), seasonal = list(order = c(0, 1, 0), period = 12)

# Fit ARIMA model using auto.arima
model2 <- auto.arima(ts_data)

# Compare models
model1
model2
```

```
> # e)
> # Manually build ARIMA model
> model1 <- Arima(ts_data, order = c(1, 0, 0), seasonal = list(order = c(0, 1, 0), period = 12))
> # Fit ARIMA model using auto.arima
> model2 <- auto.arima(ts_data)
> # Compare models
> model1
Series: ts_data
ARIMA(1,0,0)(0,1,0)[12]

Coefficients:
         ar1
      0.4830
s.e.  0.0387

sigma^2 = 10747:  log likelihood = -3090.3
AIC=6184.59   AICc=6184.62   BIC=6193.06
> model2
Series: ts_data
ARIMA(1,0,2)(0,1,2)[12] with drift

Coefficients:
         ar1      ma1      ma2     sma1     sma2   drift
      0.8458  -0.2924  -0.2814  -0.7002  -0.0857  1.0808
s.e.  0.2207   0.2586   0.1850   0.0596   0.0533  0.2055

sigma^2 = 7100:  log likelihood = -2987.44
AIC=5988.89   AICc=5989.11   BIC=6018.53
```

Figure 24: Manually building ARIMA model

(Source: Developed in R Studio)

This image shows the development of the ARIMA model (1,0,0) (0,1,0) [12] with AIC is 6184.59, AICc is 6184.62, and BIC is 6193.06.

f)

```
# f)
# Manually build ARIMA model with time trend
model3 <- Arima(ts_data, order = c(1, 0, 0), seasonal = list(order = c(0, 1, 0), period = 12)

# Fit ARIMA model with time trend using auto.arima
model4 <- auto.arima(ts_data, xreg = time(ts_data))

# Compare models
model3
model4
```

Figure 25: ARIMA model with time trend

(Source: Developed in R Studio)

ARIMA model on time trend has been represented in model 3, model 4 derived on above image.

```
> # f)
> # Manually build ARIMA model with time trend
> model3 <- Arima(ts_data, order = c(1, 0, 0), seasonal = list(order = c(0, 1, 0), period = 12), xr
eg = time(ts_data))
> # Fit ARIMA model with time trend using auto.arima
> model4 <- auto.arima(ts_data, xreg = time(ts_data))
> # Compare models
> model3
Series: ts_data
Regression with ARIMA(1,0,0)(0,1,0)[12] errors

Coefficients:
         ar1    xreg
      0.4793  9.8163
s.e.  0.0388  8.7815

sigma^2 = 10750:  log likelihood = -3089.68
AIC=6185.35   AICc=6185.4   BIC=6198.06
> model4
Series: ts_data
Regression with ARIMA(2,0,2)(0,0,2)[12] errors

Coefficients:
         ar1      ar2      ma1      ma2    sma1    sma2   intercept     xreg
      1.2801  -0.5829  -0.4396  -0.1994  0.6894  0.3825  -27470.161  14.5371
s.e.  0.0582   0.0479   0.0683   0.0592  0.0493  0.0434    2198.211   1.1020

sigma^2 = 17813:  log likelihood = -3295.42
AIC=6608.85   AICc=6609.2   BIC=6647.16
```

Figure 26: Result of the developed ARIMA model

(Source: Developed in R Studio)

The above figure shows AIC is 6608.85, AICC is 6609.2, and BIC is 6647.16.

g)

```
# g)
# Split data into training and testing sets
train_data <- window(ts_data, start = c(1973, 1), end = c(2013, 12))
test_data <- window(ts_data, start = c(2014, 1))

# Fit the models on training data
fit1 <- Arima(train_data, order = c(1, 0, 0), seasonal = list(order = c(0, 1, 0), period = 12
fit2 <- Arima(train_data, order = c(1, 0, 0), seasonal = list(order = c(0, 1, 0), period = 12

# Forecast using the models
forecast1 <- forecast(fit1, h = length(test_data))
forecast2 <- forecast(fit2, xreg = time(test_data), h = length(test_data))

# Accuracy measures
accuracy(forecast1, test_data)
accuracy(forecast2, test_data)
```

Figure 27: Splitting data into training and testing series

(Source: Developed in R Studio)

Data splitting is related into training and testing series are shown in the above figure.

```
> # g)
> # Split data into training and testing sets
> train_data <- window(ts_data, start = c(1973, 1), end = c(2013, 12))
> test_data <- window(ts_data, start = c(2014, 1))
> # Fit the models on training data
> fit1 <- Arima(train_data, order = c(1, 0, 0), seasonal = list(order = c(0, 1, 0), period = 1
2))
> fit2 <- Arima(train_data, order = c(1, 0, 0), seasonal = list(order = c(0, 1, 0), period = 1
2), xreg = time(train_data))
> # Forecast using the models
> forecast1 <- forecast(fit1, h = length(test_data))
> forecast2 <- forecast(fit2, xreg = time(test_data), h = length(test_data))
> # Accuracy measures
> accuracy(forecast1, test_data)
                      ME     RMSE       MAE        MPE     MAPE      MASE       ACF1 Theil's U
Training set    7.069339 101.4747  71.41868  0.3164788 4.415403 0.8690035 0.03014959        NA
Test set      -49.203723 154.2995 101.31103 -3.1525932 5.564103 1.2327256 0.35292565 0.5097313
> accuracy(forecast2, test_data)
                      ME     RMSE       MAE        MPE     MAPE      MASE       ACF1 Theil's U
Training set  -0.4663606 101.3065  71.87483 -0.2199657 4.468001 0.8745539 0.03781773        NA
Test set     -71.1006274 166.3280 112.87978 -4.5004680 6.321429 1.3734911 0.38789843 0.5538807
```

Figure 28: Result of Splitting data into training and testing series

(Source: Developed in R Studio)

The output of the training set is 0.4663606, and the test set is -71.1006274 shown in the image.

h)

```
# h)
# Fit the final model on the full dataset
final_model <- Arima(ts_data, order = c(1, 0, 0), seasonal = list(order = c(0, 1, 0), period = 12))

# Forecast for 24 periods
forecast_final <- forecast(final_model, h = 24)

# Plot the forecast
plot(forecast_final, main = "Energy Consumption Forecast", xlab = "Year", ylab = "Energy (MW)")
```

Figure 29: Fit the final model on the full dataset

(Source: Developed in R Studio)

The final model has been set on the full dataset derived here.

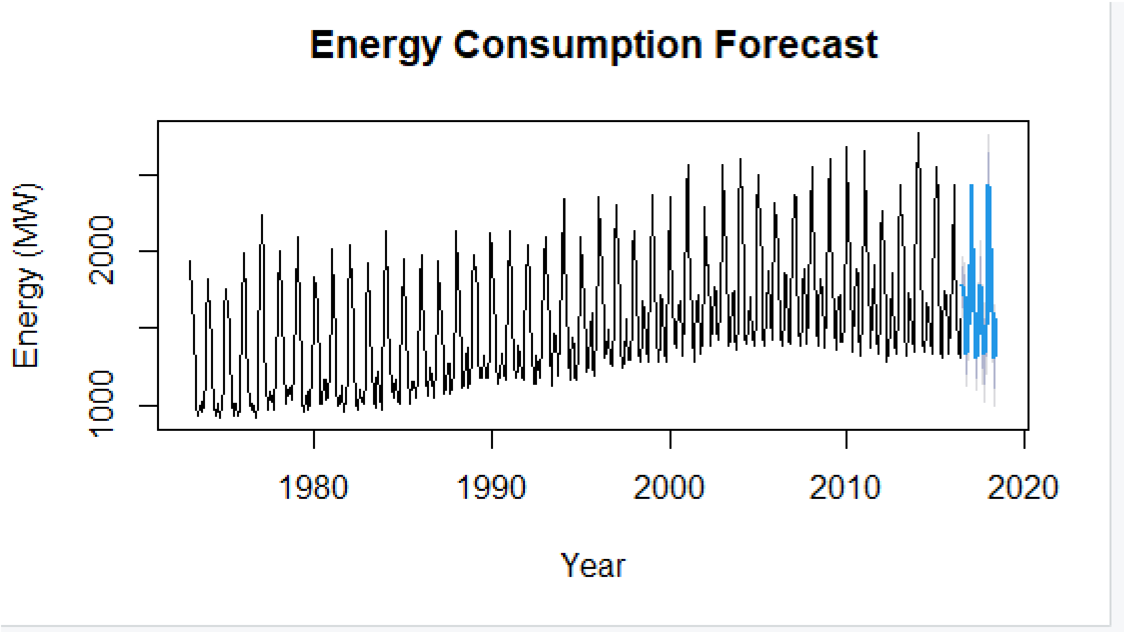


Figure 30: Plotting energy consumption forecast

(Source: Developed in R Studio)

The energy consumption plot has been done in R studio.

Problem 3

```r
# Problem 3

# Load the data
# Load the dataset
data <- read.table("steel2.txt", header = TRUE)

# Convert to time series
ts_data <- ts(data$X5980, start = 1960, end = 2019)
library(forecast)

# Fit ARIMA model
arima_model <- auto.arima(ts_data)

# Test coefficients for significance
coeftest(arima_model)

# Analyze residuals for autocorrelation
Box.test(arima_model$residuals)
# Mosel Building
arima_model
auto_arima_model <- forecast::auto.arima(ts_data)
auto_arima_model
# Backtest comparison
accuracy(arima_model)
accuracy(auto_arima_model)

# 10-year forecast
forecast_arima <- forecast(arima_model, h = 10)
forecast_auto_arima <- forecast(auto_arima_model, h = 10)

# Plot forecasts
plot(forecast_arima, main = "ARIMA Forecast  (Next 10 years)")
plot(forecast_auto_arima, main = "Auto ARIMA Forecast  (Next 10 years)")
```

Figure 31: Dataset loading

(Source: Developed in R Studio)

The dataset has been loaded here derived in the above image.

```
> # Mosel Building
> arima_model
Series: ts_data
ARIMA(2,1,1)

Coefficients:
          ar1      ar2     ma1
      -1.2849  -0.6232  0.8206
s.e.   0.1222   0.1063  0.0926

sigma^2 = 171654:  log likelihood = -438.28
AIC=884.57   AICc=885.31   BIC=892.88
> auto_arima_model <- forecast::auto.arima(ts_data)
> auto_arima_model
Series: ts_data
ARIMA(2,1,1)

Coefficients:
          ar1      ar2     ma1
      -1.2849  -0.6232  0.8206
s.e.   0.1222   0.1063  0.0926

sigma^2 = 171654:  log likelihood = -438.28
AIC=884.57   AICc=885.31   BIC=892.88
> # Backtest comparison
> accuracy(arima_model)
                   ME    RMSE      MAE          MPE    MAPE      MASE       ACF1
Training set 19.02117 400.263 338.9269 -0.007878093 5.49793 0.8081429 0.06208639
> accuracy(auto_arima_model)
                   ME    RMSE      MAE          MPE    MAPE      MASE       ACF1
Training set 19.02117 400.263 338.9269 -0.007878093 5.49793 0.8081429 0.06208639
```

Figure 32: Output of ARIMA model

(Source: Developed in R Studio)

The above figure depicts the training set as ME 19.02117, RMSE 400.263.

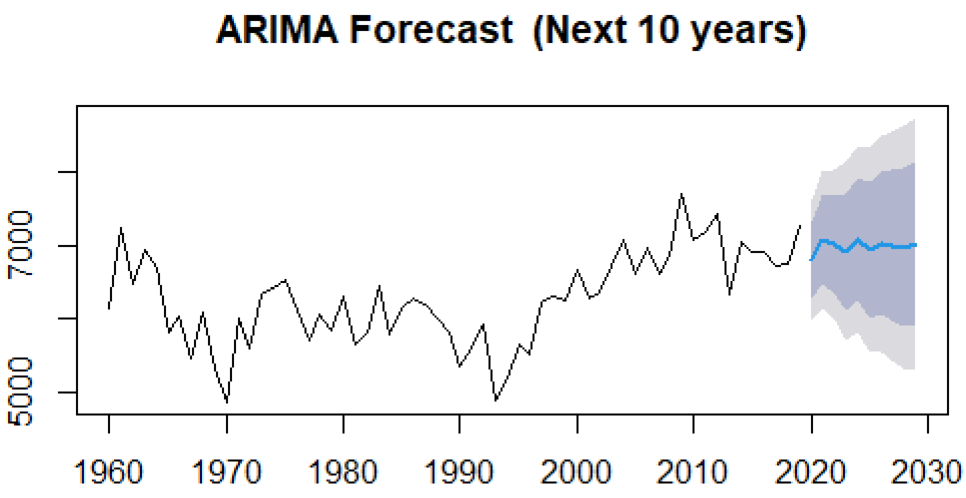


Figure 33: ARIMA forecasting in the next 10 years

(Source: Developed in R Studio)

This is to forecast using ARIMA over the next ten years in R studio.

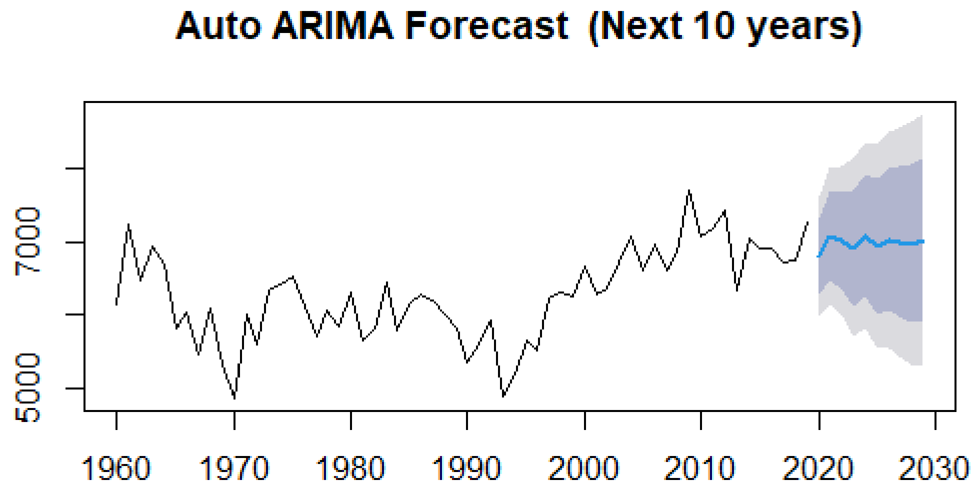


Figure 34: Auto ARIMA forecasting in the next 10 years

(Source: Developed in R Studio)

Auto ARIMA model forecasting is done over the next ten years.

Problem 4

a)

```r
# Problem 4
# a)
# Load the data
groceries <- read.csv("groceries (1).csv")

# Convert the Date column to a proper date format
groceries$Date <- as.Date(groceries$Date, format = "%d-%b-%y")

# Create time series for each product
toothpaste <- ts(groceries$ToothPaste, start = c(2008, 1), frequency = 52)
peanut_butter <- ts(groceries$PeanutButter, start = c(2008, 1), frequency = 52)
biscuits <- ts(groceries$Biscuits, start = c(2008, 1), frequency = 52)

# Plot the time series
plot(toothpaste, main = "ToothPaste Sales")
plot(peanut_butter, main = "Peanut Butter Sales")
plot(biscuits, main = "Biscuits Sales")
```

Figure 35: Loading data and converting data column

(Source: Developed in R Studio)

Data loading and data conversion for a column is developed in the above image.

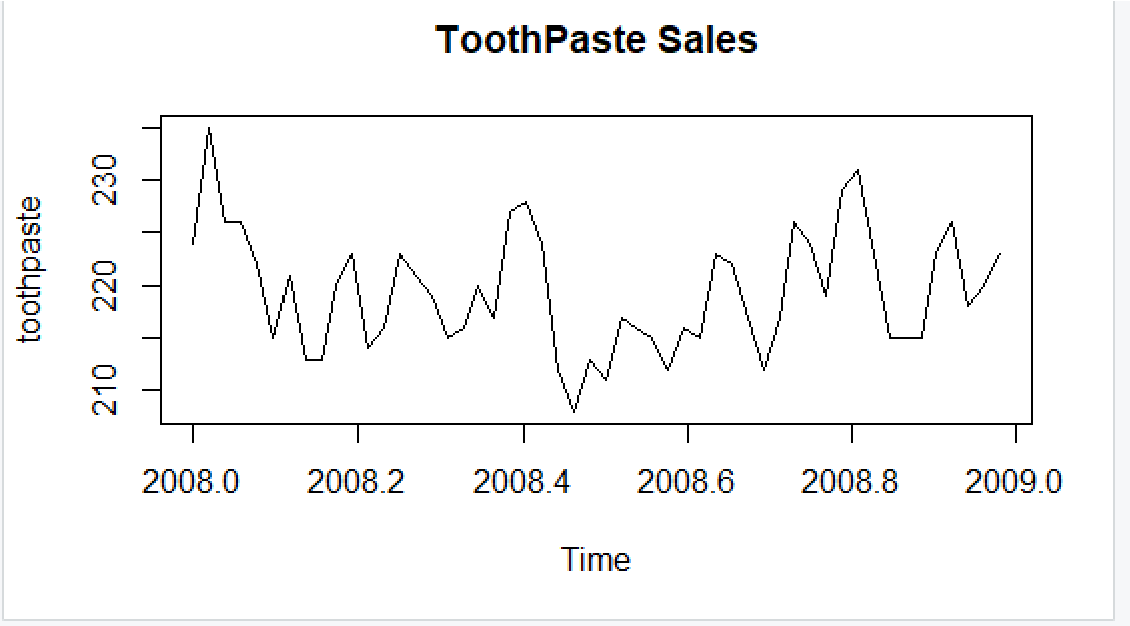


Figure 35: Plotting of toothpaste sales

(Source: Developed in R Studio)

Plotting of toothpaste sales is done in the above image generated by R studio.

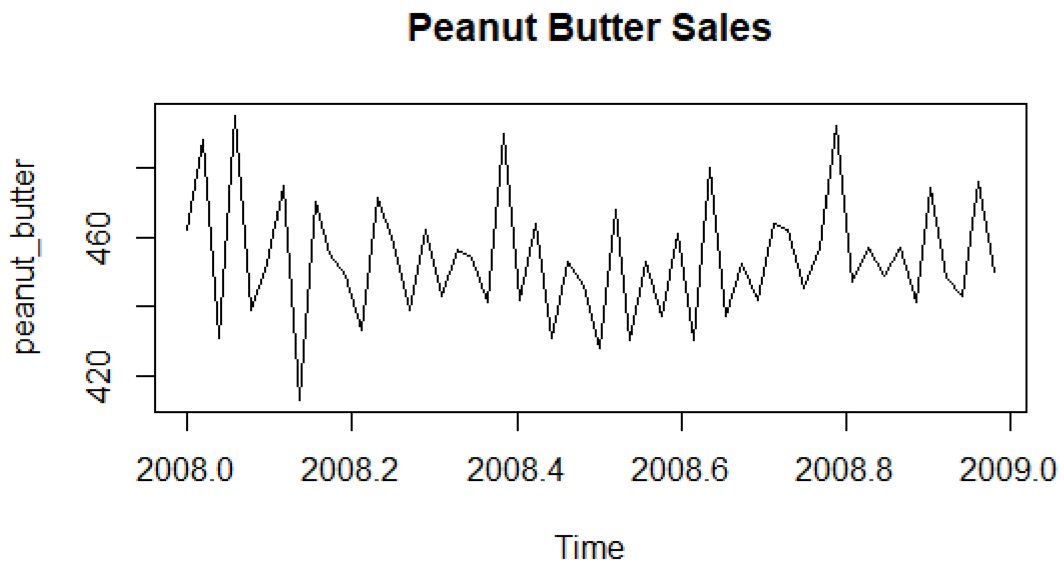


Figure 35: Plotting of Peanut Butter sales

(Source: Developed in R Studio)

Plotting of peanut butter sales can be represented in the above figure.

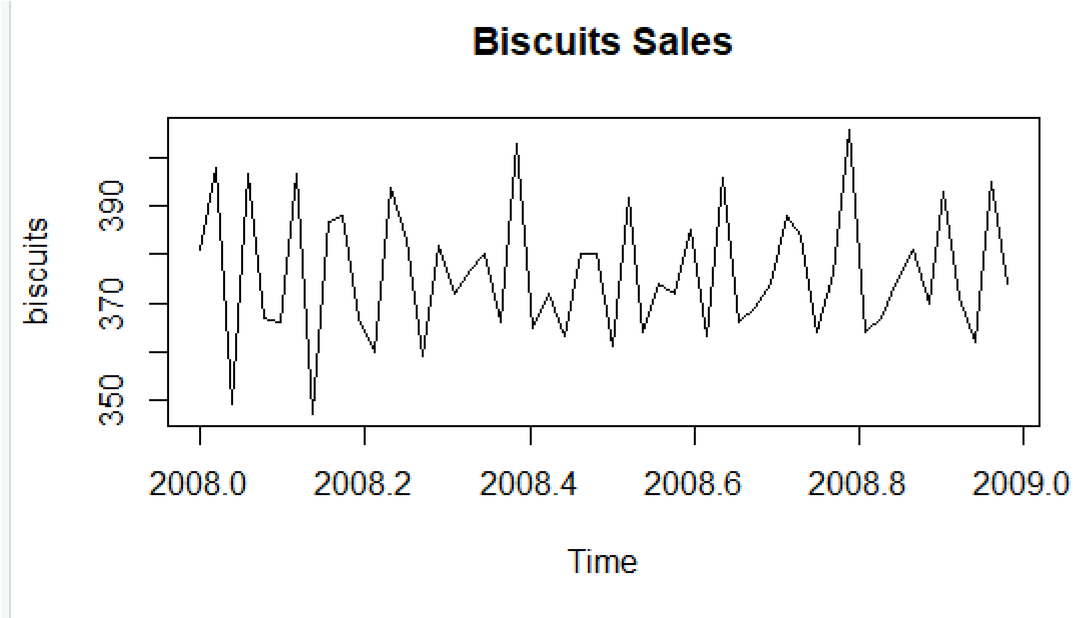


Figure 36: Plotting of Biscuit sales

(Source: Developed in R Studio)

The above figure represents the plotting of biscuit sales in R Studio.

b)

```
# b)
# Create lag plots
lag.plot(toothpaste, lag.max = 12, main = "Lag Plot: ToothPaste")
lag.plot(peanut_butter, lag.max = 12, main = "Lag Plot: Peanut Butter")
lag.plot(biscuits, lag.max = 12, main = "Lag Plot: Biscuits")

# Create cross-correlation plots
ccf(toothpaste, peanut_butter, lag.max = 12, main = "Cross-Correlation: ToothPaste vs Peanut Butter")
ccf(toothpaste, biscuits, lag.max = 12, main = "Cross-Correlation: ToothPaste vs Biscuits")
```

Figure 36: Create lag plots

(Source: Developed in R Studio)

The created lag plots can be depicted here in the above image.

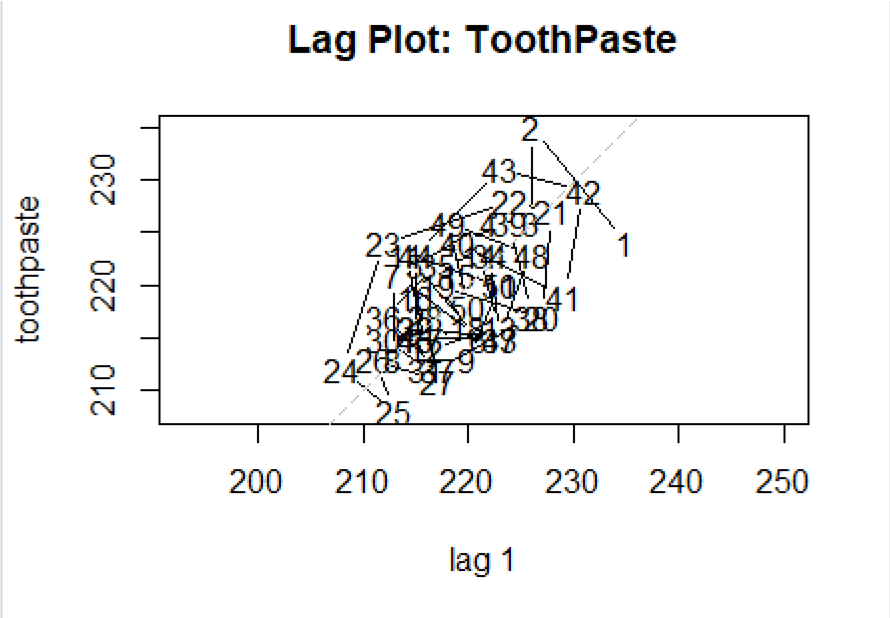


Figure 37: Plotting of lag plot and toothpaste sales

(Source: Developed in R Studio)

Lag plot and toothpaste sales analysis is done in the above image.

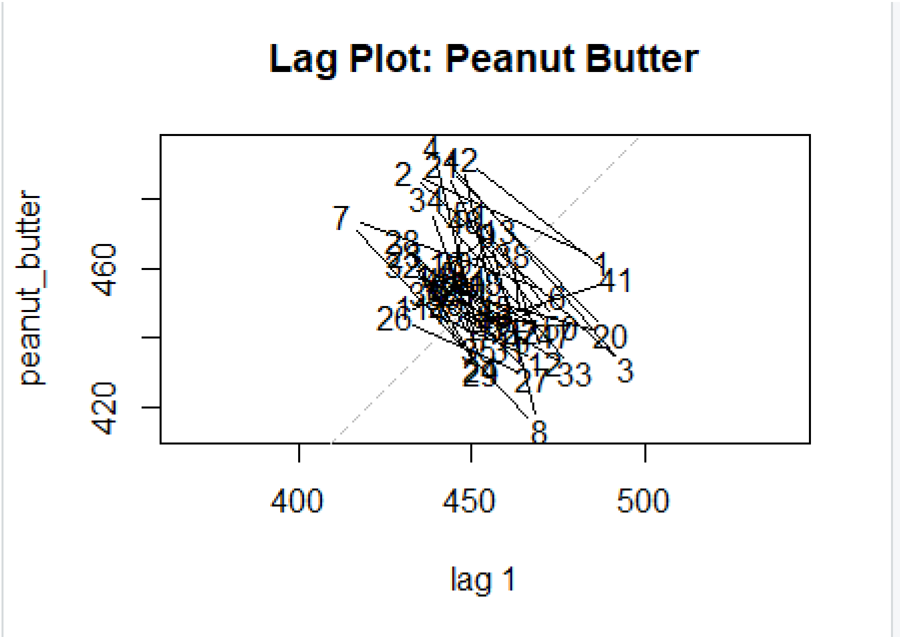


Figure 38: Plotting of lag plot and peanut butter

(Source: Developed in R Studio)

Lag plot and peanut butter plotting are done here in a resultant way.

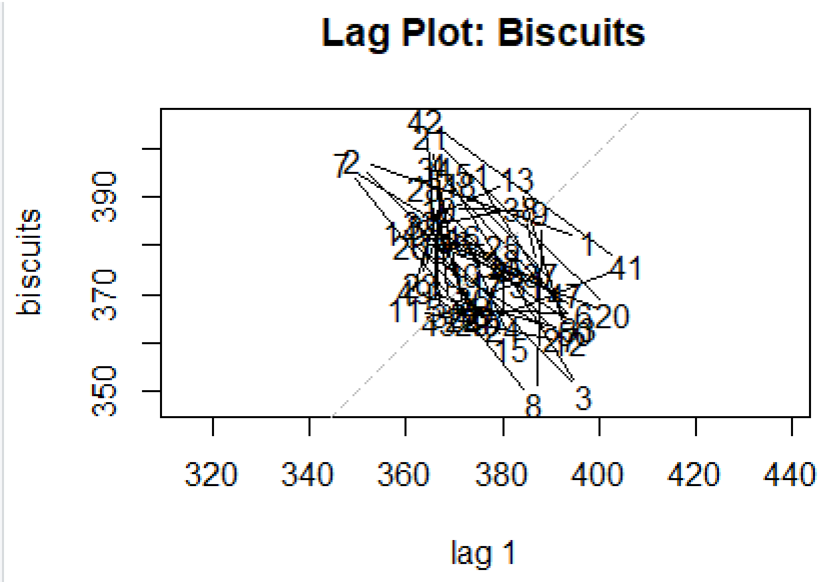


Figure 39: Plotting of lag plot and biscuits

(Source: Developed in R Studio)

Plotting of lag plot and biscuits can be depicted here in R studio.

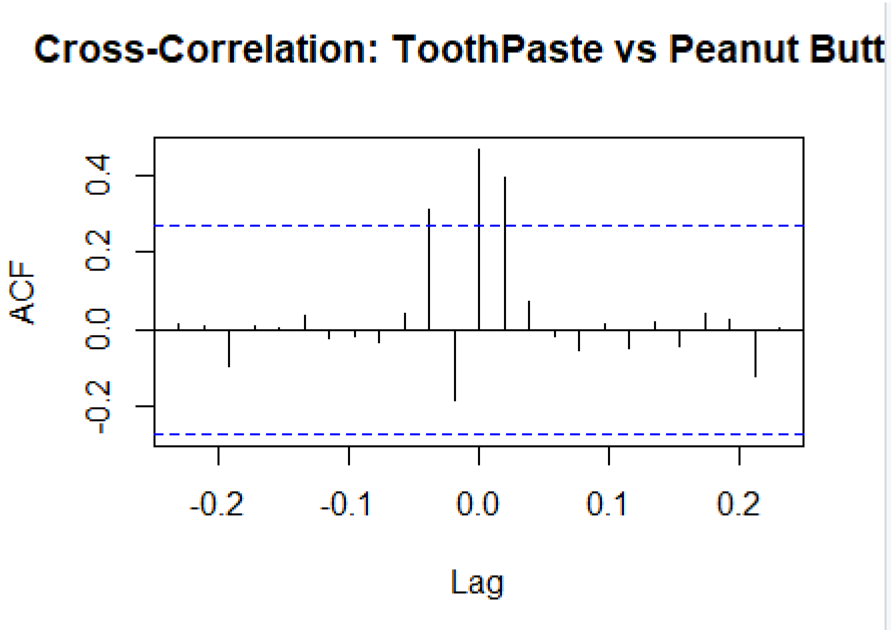


Figure 40: Plotting of cross-correlation of lag plot and peanut butter

(Source: Developed in R Studio)

lag plot cross-correlation with peanut butter plotting has been done in R studio software.

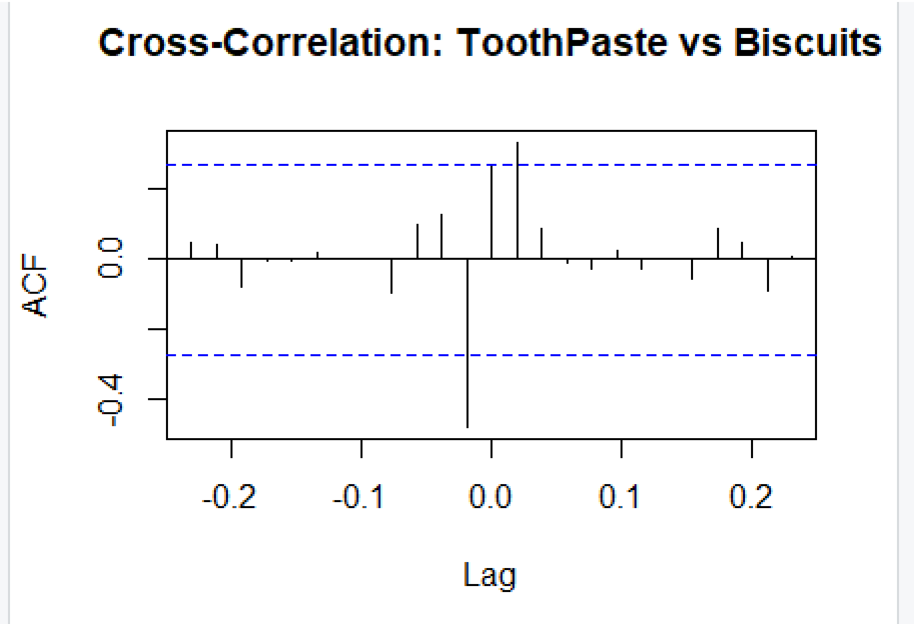
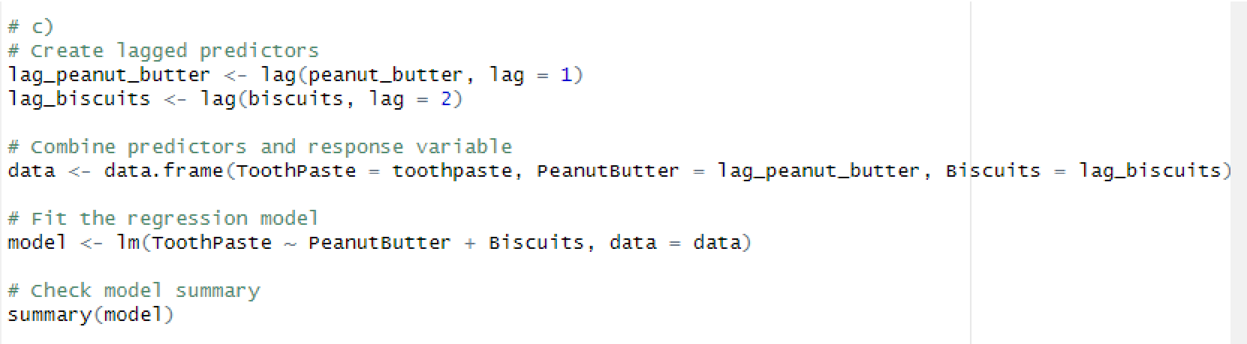


Figure 41: Plotting of cross-correlation of Toothpaste vs Biscuit

(Source: Developed in R Studio)

Plotting the cross-correlation between biscuits and toothpaste is depicted in the image.

c)



```r
# c)
# Create lagged predictors
lag_peanut_butter <- lag(peanut_butter, lag = 1)
lag_biscuits <- lag(biscuits, lag = 2)

# Combine predictors and response variable
data <- data.frame(ToothPaste = toothpaste, PeanutButter = lag_peanut_butter, Biscuits = lag_biscuits)

# Fit the regression model
model <- lm(ToothPaste ~ PeanutButter + Biscuits, data = data)

# Check model summary
summary(model)
```

Figure 42: Create lagged predictors

(Source: Developed in R Studio)

This figure shows the create lagged predictors on the basis of the regression model.

```
> # c)
> # Create lagged predictors
> lag_peanut_butter <- lag(peanut_butter, lag = 1)
> lag_biscuits <- lag(biscuits, lag = 2)
> # Combine predictors and response variable
> data <- data.frame(ToothPaste = toothpaste, PeanutButter = lag_peanut_butter, Biscuits = lag_biscuits)
> # Fit the regression model
> model <- lm(ToothPaste ~ PeanutButter + Biscuits, data = data)
> # Check model summary
> summary(model)

Call:
lm(formula = ToothPaste ~ PeanutButter + Biscuits, data = data)

Residuals:
   Min     1Q Median     3Q    Max 
-9.399 -2.897 -1.186  2.843  9.173 

Coefficients:
              Estimate Std. Error t value Pr(>|t|)    
(Intercept)  175.81786   17.45111  10.075 1.57e-13 ***
PeanutButter   0.47691    0.09258   5.151 4.59e-06 ***
Biscuits      -0.45910    0.12019  -3.820 0.000377 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 4.517 on 49 degrees of freedom
Multiple R-squared:  0.398,	Adjusted R-squared:  0.3734 
F-statistic:  16.2 on 2 and 49 DF,  p-value: 3.98e-06

>
```

Figure 43: Output of creating lagged predictors

(Source: Developed in R Studio)

According to the above figure the result of predictors by lagged creation.

d)

```
# d)
# Install and load the forecast package
library(forecast)

# Combine lagged predictors and response variable
arima_data <- data.frame(ToothPaste = toothpaste, PeanutButter = lag_peanut_butter, Biscuits = lag_bis

# Convert predictors to a numeric matrix
xreg <- as.matrix(arima_data[, c("PeanutButter", "Biscuits")])

# Fit auto.arima model
arima_model <- auto.arima(arima_data[, "ToothPaste"], xreg = xreg)

# Print the auto.arima model summary
summary(arima_model)


# Print the auto.arima model summary
summary(arima_model)
```

Figure 44: Installing and loading the forecast package

(Source: Developed in R Studio)

The above figure using for the installation and loading of the forecast package.

```
> # Fit auto.arima model
> arima_model <- auto.arima(arima_data[, "ToothPaste"], xreg = xreg)
> # Print the auto.arima model summary
> summary(arima_model)
Series: arima_data[, "ToothPaste"]
Regression with ARIMA(3,0,0) errors

Coefficients:
         ar1      ar2     ar3  intercept  PeanutButter  Biscuits
      0.9672  -0.7479  0.3642   173.4805       -0.0221    0.1493
s.e.  0.1479   0.1764  0.1443     8.2614        0.0575    0.0814

sigma^2 = 16.64:  log likelihood = -144.34
AIC=302.68   AICc=305.22   BIC=316.33

Training set error measures:
                      ME     RMSE      MAE         MPE     MAPE MASE      ACF1
Training set -0.02745298 3.837139 3.195449 -0.05075112 1.453755  NaN 0.1164065
> # Print the auto.arima model summary
> summary(arima_model)
Series: arima_data[, "ToothPaste"]
Regression with ARIMA(3,0,0) errors

Coefficients:
         ar1      ar2     ar3  intercept  PeanutButter  Biscuits
      0.9672  -0.7479  0.3642   173.4805       -0.0221    0.1493
s.e.  0.1479   0.1764  0.1443     8.2614        0.0575    0.0814

sigma^2 = 16.64:  log likelihood = -144.34
AIC=302.68   AICc=305.22   BIC=316.33

Training set error measures:
                      ME     RMSE      MAE         MPE     MAPE MASE      ACF1
Training set -0.02745298 3.837139 3.195449 -0.05075112 1.453755  NaN 0.1164065
> 
```

Figure 45: Fit auto. Arima model

(Source: Developed in R Studio)

In this figure, using the model of Fit Auto. Arima.

Problem 5

a)

```
# Problem 5
#a)
# Load required libraries
library(rugarch)

# Read the data
amzn_data <- read.csv("amzn_2005_13_d.csv", header = TRUE)
amzn_data$Date <- as.Date(amzn_data$Date, format = "%m/%d/%Y")

# Compute log returns
amzn_data$log_returns <- c(NA, diff(log(amzn_data$Price)))
```

Figure 46: Loading required libraries

(Source: Developed in R Studio)

In this specific figure, the required libraries are loading.

	Date	Price	log_returns
1	2005-01-03	44.52	NA
2	2005-01-04	42.14	-0.0549411180
3	2005-01-05	41.77	-0.0088190299
4	2005-01-06	41.05	-0.0173875426
5	2005-01-07	42.32	0.0304689517
6	2005-01-10	41.84	-0.0114069678
7	2005-01-11	41.64	-0.0047915760
8	2005-01-12	42.30	0.0157258423
9	2005-01-13	42.60	0.0070671672
10	2005-01-14	44.55	0.0447579006
11	2005-01-18	44.58	0.0006731740
12	2005-01-19	43.96	-0.0140051984
13	2005-01-20	42.36	-0.0370756088
14	2005-01-21	41.16	-0.0287376098
15	2005-01-24	40.38	-0.0191322981
16	2005-01-25	40.94	0.0137729673
17	2005-01-26	41.34	0.0097229740
18	2005-01-27	42.31	0.0231929105
19	2005-01-28	42.22	-0.0021294223
20	2005-01-31	43.22	0.0234093087
21	2005-02-01	42.48	-0.0172699741

Showing 1 to 22 of 2,061 entries, 3 total columns

Figure 47: Showing entries and columns

(Source: Developed in R Studio)

This picture shows some entries and columns like date, price, and log_returns

b)

```
# b)
# Autocorrelation analysis
acf(amzn_data$log_returns[-1], lag.max = 20, main = "Autocorrelation of Log Returns")
```

Figure 48: Autocorrelation analysis

(Source: Developed in R Studio)

The above picture shows the analysis of autocorrelation.

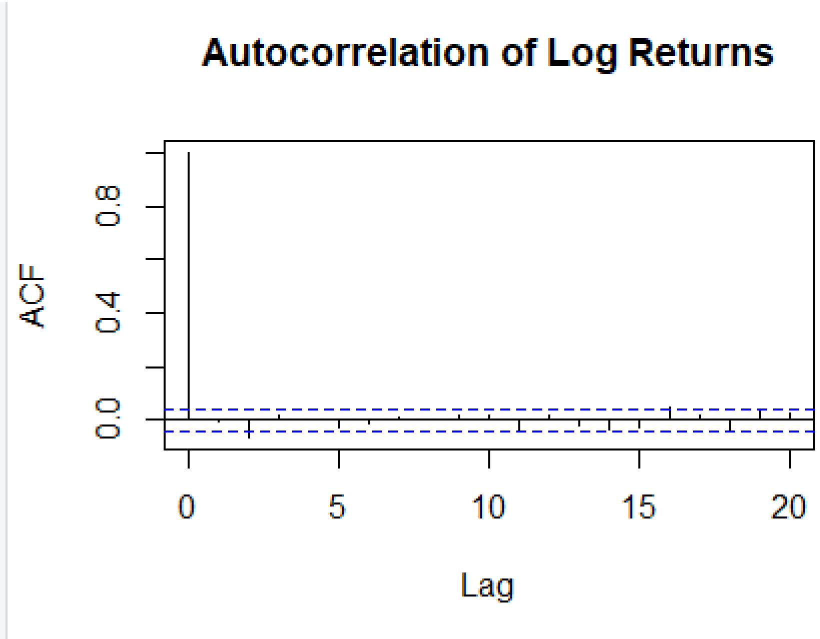


Figure 49: Plot of autocorrelation of log returns

(Source: Developed in R Studio)

After analysis of autocorrelation, then this picture shows the plot of autocorrelation of log returns.

c)

```
# c)
# ARCH effects test (Box-Ljung test)
box_test <- Box.test(amzn_data$log_returns[-1]^2, lag = 20, type = "Ljung-Box")
arch_test_result <- ifelse(box_test$p.value <= 0.05, "Significant ARCH effects", "No significant ARCH

# Print ARCH effects test result
print(arch_test_result)
```

Figure 50: ARCH effects test

(Source: Developed in R Studio)

In the above picture, test the effects of ARCH.

```
box_test          List of  5
   $ statistic: Named num 54.4
    ..- attr(*, "names")= chr "X-squared"
   $ parameter: Named num 20
    ..- attr(*, "names")= chr "df"
   $ p.value  : num 5.12e-05
   $ method   : chr "Box-Ljung test"
   $ data.name: chr "amzn_data$log_returns[-1]^2"
   - attr(*, "class")= chr "htest"
Values
  arch_test_re…   "Significant ARCH effects"
```

Figure 51: Box test

(Source: Developed in R Studio)

This specific figure shows the test of the box and which was developed by R Studio.

d)

```
# d)
# Fit ARMA-GARCH model
spec <- ugarchspec(variance.model = list(model = "sGARCH", garchOrder = c(1, 1)),
                   mean.model = list(armaOrder = c(1, 0)))
fit <- ugarchfit(spec, amzn_data$log_returns[-1])

# Model checking
residuals <- fit@fit$residuals
sigma <- fit@fit$sigma
tsdisplay(residuals, main = "Residuals")
tsdisplay(residuals^2, main = "Squared Residuals")
plot(density(residuals, na.rm = TRUE), main = "Density of Residuals")
```

Figure 52: Fit ARMA-GARCH model

(Source: Developed in R Studio)

Developing by R studio, shows the model of Fit ARMA-GARCH.

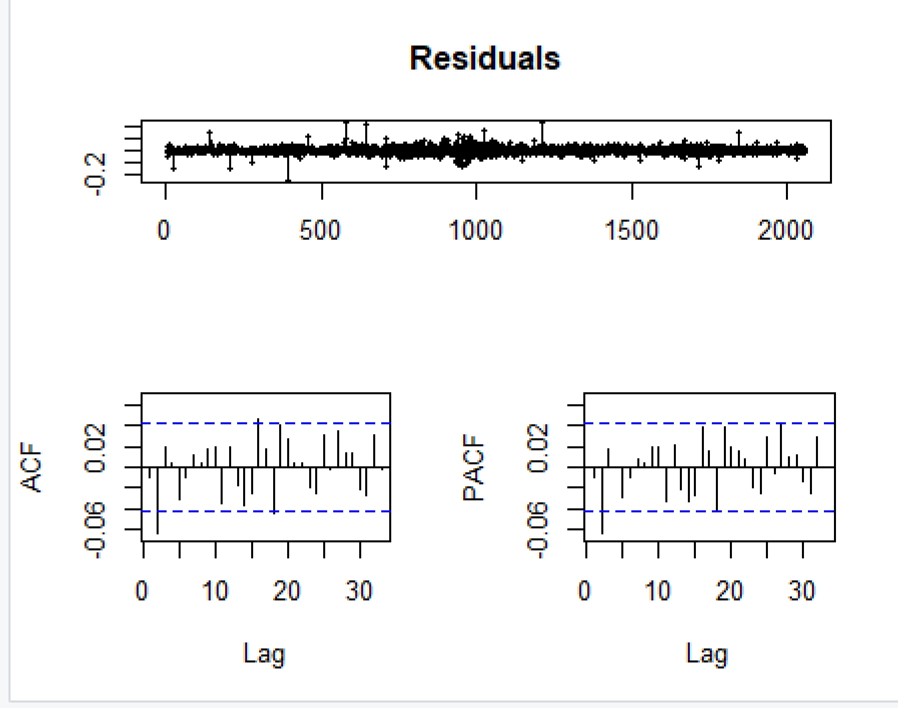


Figure 53: Plot of residuals of ACF and PACF

(Source: Developed in R Studio)

This picture shows the plot residual of ACF and PACF.

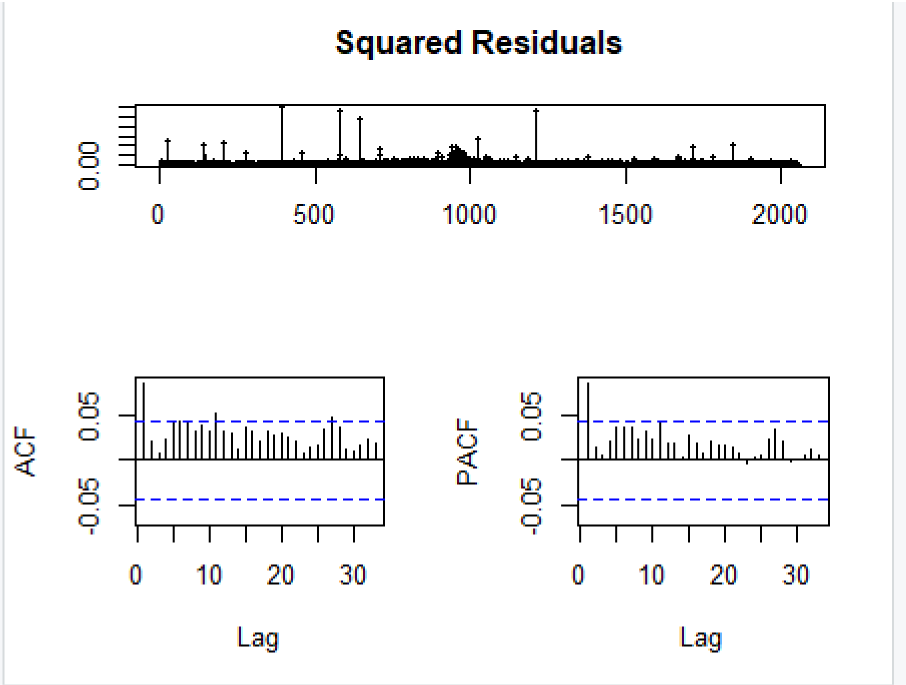


Figure 54: Squared residual plot

(Source: Developed in R Studio)

Using R studio, this picture shows the squared residual plot.

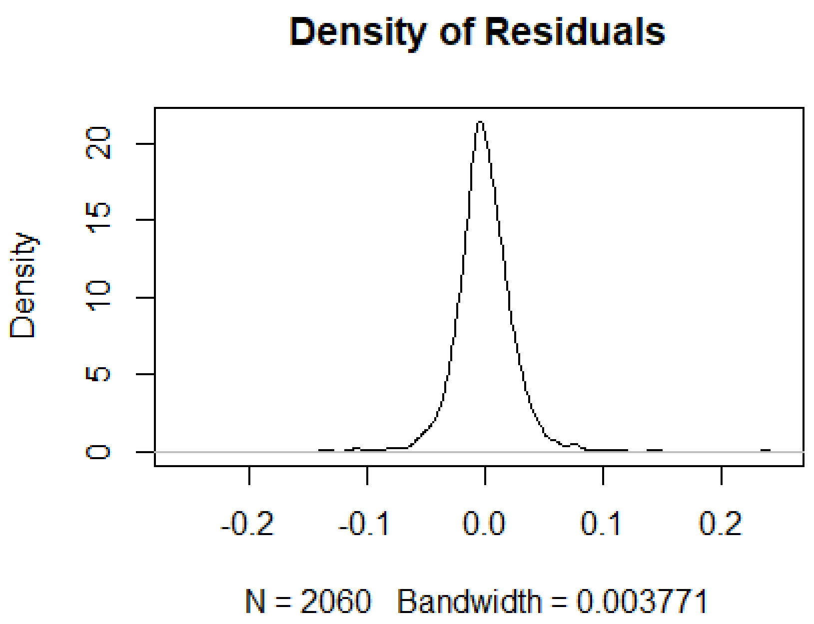


Figure 55: Density of residual plots

(Source: Developed in R Studio)

Using R studio, this figure shows the destiny of residual plots.

e)

```
# e)
# Fitted GARCH(1,1) model
fit

# Interpretation of alpha1 and beta1 parameters
estimated_alpha1 <- fit@fit$coef[2]
estimated_beta1 <- fit@fit$coef[3]
interpretation <- paste("The estimated alpha1 parameter is", round(estimated_alpha1, 4),
                        "and the estimated beta1 parameter is", round(estimated_beta1, 4))
interpretation
```

Figure 56: Fitted GARCH model

(Source: Developed in R Studio)

This picture shows the fitted GARCH model implemented by R Studio.

```
> # e)
> # Fitted GARCH(1,1) model
> fit

*---------------------------------*
*          GARCH Model Fit        *
*---------------------------------*

Conditional Variance Dynamics
-----------------------------------
GARCH Model     : sGARCH(1,1)
Mean Model      : ARFIMA(1,0,0)
Distribution    : norm

Optimal Parameters
------------------------------------
        Estimate  Std. Error  t value Pr(>|t|)
mu      0.001042    0.000586   1.77812 0.075384
ar1     0.006273    0.023927   0.26217 0.793188
omega   0.000017    0.000002  10.47064 0.000000
alpha1  0.016315    0.002269   7.18899 0.000000
beta1   0.960406    0.003240 296.43541 0.000000

Robust Standard Errors:
        Estimate  Std. Error  t value Pr(>|t|)
mu      0.001042    0.000524   1.98922 0.046677
ar1     0.006273    0.025262   0.24832 0.803889
omega   0.000017    0.000002   7.97787 0.000000
alpha1  0.016315    0.005226   3.12192 0.001797
beta1   0.960406    0.005738 167.36561 0.000000

LogLikelihood : 4518.134
```

Figure 57: Result of the GARCH model

(Source: Developed in R Studio)

After implementing the GARCH model, the above picture shows the outcome of the GARCH model.

```
Nyblom stability test
------------------------------------
Joint Statistic:  12.4836
Individual Statistics:
mu     0.1599
ar1    0.3347
omega  0.7385
alpha1 0.3387
beta1  0.2769

Asymptotic Critical Values (10% 5% 1%)
Joint Statistic:        1.28 1.47 1.88
Individual Statistic:   0.35 0.47 0.75

Sign Bias Test
------------------------------------
                   t-value   prob sig
Sign Bias            1.435 0.1515
Negative Sign Bias   1.095 0.2735
Positive Sign Bias   1.264 0.2063
Joint Effect         6.024 0.1104


Adjusted Pearson Goodness-of-Fit Test:
------------------------------------
  group statistic p-value(g-1)
1    20     213.3    7.502e-35
2    30     225.0    3.370e-32
3    40     247.3    4.283e-32
4    50     256.6    6.580e-30


Elapsed time : 0.352922
```

Figure 58: Nyblom stability test

(Source: Developed in R Studio)

Helping by R studio, in this picture doing the Nyblom stability test.

```
> # Interpretation of alpha1 and beta1 parameters
> estimated_alpha1 <- fit@fit$coef[2]
> estimated_beta1 <- fit@fit$coef[3]
> interpretation <- paste("The estimated alpha1 parameter is", round(estimated_alpha1, 4),
+                         "and the estimated beta1 parameter is", round(estimated_beta1, 4))
> interpretation
[1] "The estimated alpha1 parameter is 0.0063 and the estimated beta1 parameter is 0"
```

Figure 59: Interpretation of alpha and beta parameters

(Source: Developed in R Studio)

f)

```
# f)
# 5-step ahead forecasts of stock volatility
forecast <- ugarchforecast(fit, n.ahead = 5)
forecast@forecast$seriesFor
```

Figure 60: 5-step ahead forecasts of stock volatility

(Source: Developed in R Studio)

By helping R Studio, the above figure shows the stock volatility of the 5-step ahead forecast.

```
> # f)
> # 5-step ahead forecasts of stock volatility
> forecast <- ugarchforecast(fit, n.ahead = 5)
> forecast@forecast$seriesFor
    1975-08-23 05:30:00
T+1         0.001101598
T+2         0.001042024
T+3         0.001041650
T+4         0.001041648
T+5         0.001041648
>
```

Figure 61: 5-step ahead forecasts

(Source: Developed in R Studio)

The above figure is implemented by R studio and shows the forecasts of 5 steps ahead.