ECON 144 HW 2

Leonard Zhu

2024-10-25

Problem 4.6 (i.e., Chapter 4, Problem 6) from Textbook (a)

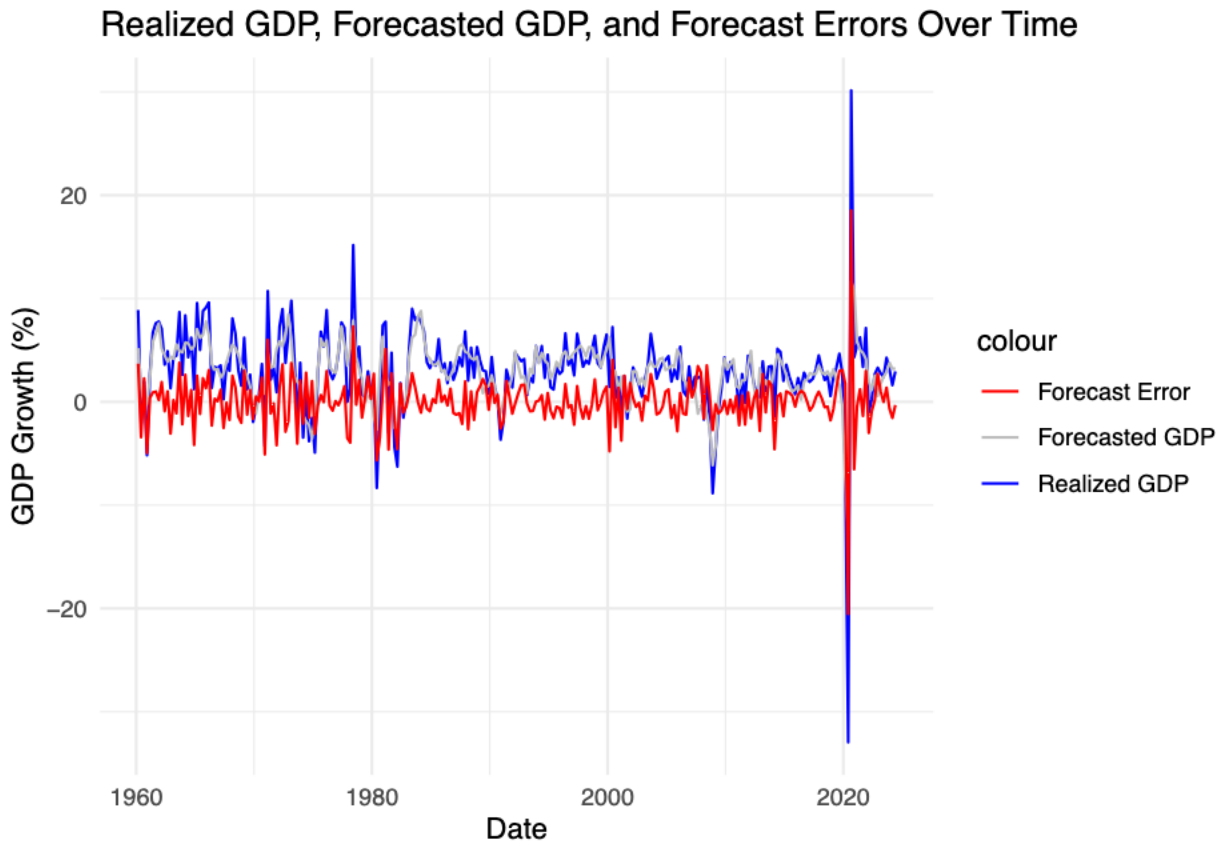
```
gdp_data <- read_excel("~/Downloads/gdpplus.xlsx")
```

```
# Convert year and quarter to date format for better plotting
gdp_data <- gdp_data %>%
  mutate(Date = as.Date(paste(OBS_YEAR, OBS_QUARTER * 3, 1, sep = "-"), "%Y-%m-%d"))

# Rename columns for clarity
gdp_data <- gdp_data %>%
  rename(Realized_GDP = GRGDP_DATA, Forecasted_GDP = GDPPLUS_DATA) # Adjust names as needed
```

```
# Calculate forecast error
gdp_data <- gdp_data %>%
  mutate(Forecast_Error = Realized_GDP - Forecasted_GDP)
head(gdp_data)
```

```
## # A tibble: 6 x 9
##   OBS_YEAR OBS_QUARTER OBS_QUARTER2 RECBARS Realized_GDP GRGDI_DATA
##      <dbl>       <dbl>        <dbl>   <dbl>        <dbl>      <dbl>
## 1     1960           1         0          0         8.90     8.80
## 2     1960           2         0.25       1        -2.17    -0.0683
## 3     1960           3         0.5        1         1.96     0.0228
## 4     1960           4         0.75       1        -5.16    -2.18
## 5     1961           1         0          1         2.69     0.812
## 6     1961           2         0.25       0         6.73     6.70
## # i 3 more variables: Forecasted_GDP <dbl>, Date <date>, Forecast_Error <dbl>
```

```
ggplot(gdp_data, aes(x = Date)) +
  geom_line(aes(y = Realized_GDP, color = "Realized GDP")) +
  geom_line(aes(y = Forecasted_GDP, color = "Forecasted GDP")) +
  geom_line(aes(y = Forecast_Error, color = "Forecast Error")) +
  labs(title = "Realized GDP, Forecasted GDP, and Forecast Errors Over Time",
       x = "Date", y = "GDP Growth (%)") +
  scale_color_manual(values = c("Realized GDP" = "blue",
                                "Forecasted GDP" = "grey",
                                "Forecast Error" = "red")) +
  theme_minimal()
```




```
# Descriptive statistics
summary_stats <- gdp_data %>%
  summarise(Realized_GDP_Mean = mean(Realized_GDP, na.rm = TRUE),
            Realized_GDP_SD = sd(Realized_GDP, na.rm = TRUE),
            Forecasted_GDP_Mean = mean(Forecasted_GDP, na.rm = TRUE),
            Forecasted_GDP_SD = sd(Forecasted_GDP, na.rm = TRUE),
            Forecast_Error_Mean = mean(Forecast_Error, na.rm = TRUE),
            Forecast_Error_SD = sd(Forecast_Error, na.rm = TRUE))
summary_stats
```

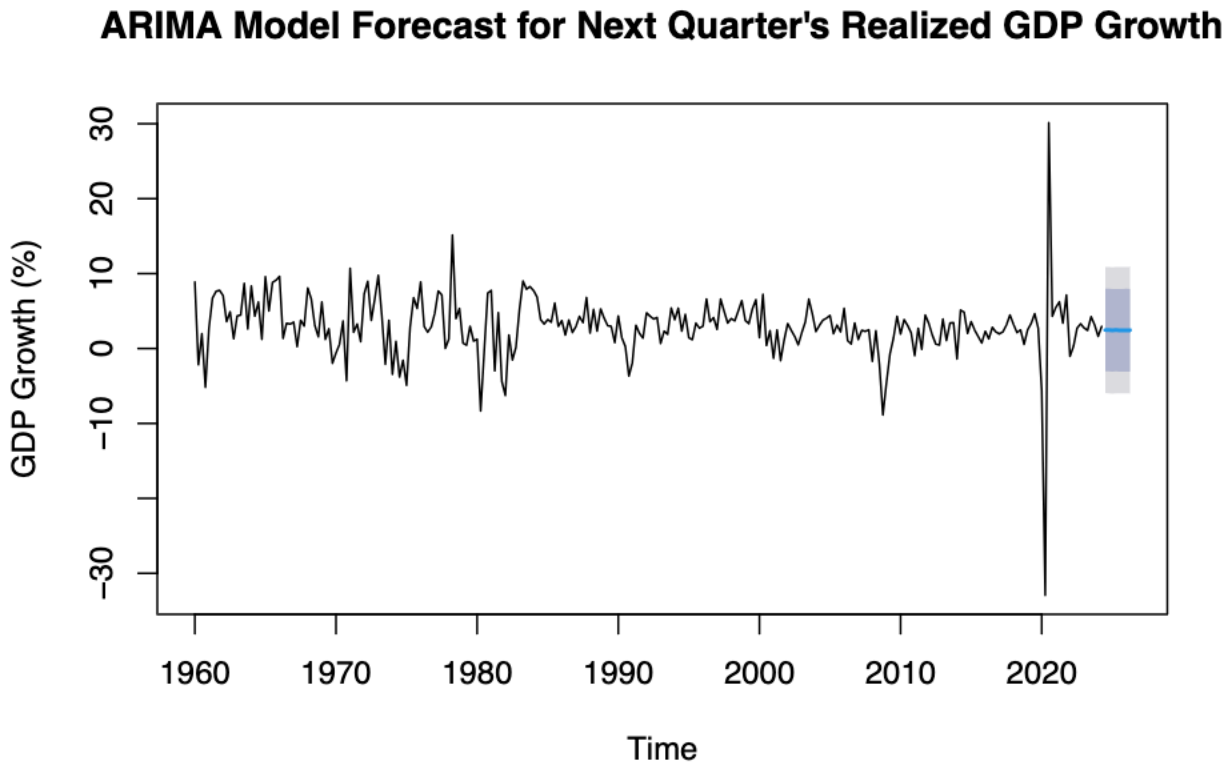
```
## # A tibble: 1 x 6
##   Realized_GDP_Mean Realized_GDP_SD Forecasted_GDP_Mean Forecasted_GDP_SD
##               <dbl>           <dbl>               <dbl>             <dbl>
## 1              2.96            4.27                2.95              2.70
## # i 2 more variables: Forecast_Error_Mean <dbl>, Forecast_Error_SD <dbl>
```

```
# Load necessary libraries
library(forecast)

# Fit an ARIMA model to the Realized GDP data
gdp_ts <- ts(gdp_data$Realized_GDP, frequency = 4, start = c(min(gdp_data$OBS_YEAR), min(gdp_data$OBS_Q

# Fit the ARIMA model
arima_model <- auto.arima(gdp_ts)
```

```r
# Forecast the next quarter
gdp_forecast <- forecast(arima_model, h = )

# Print the forecast results
print(gdp_forecast)
```

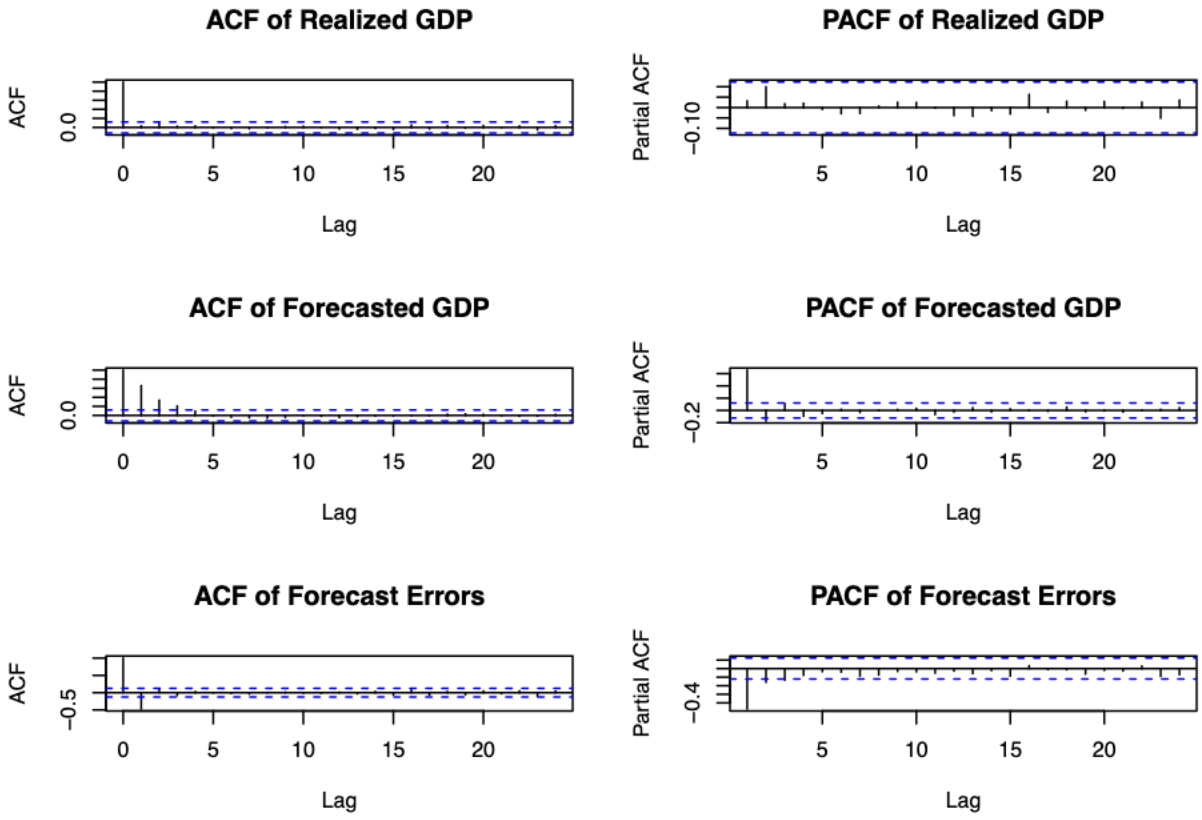
```
##          Point Forecast     Lo 80    Hi 80     Lo 95    Hi 95
## 2024 Q3       2.472186 -3.038623 7.982994 -5.955868 10.90024
## 2024 Q4       2.456441 -3.055272 7.968155 -5.972997 10.88588
## 2025 Q1       2.425967 -3.086651 7.938586 -6.004855 10.85679
## 2025 Q2       2.450283 -3.063240 7.963807 -5.981923 10.88249
## 2025 Q3       2.439361 -3.077629 7.956352 -5.998148 10.87687
## 2025 Q4       2.442161 -3.075765 7.960088 -5.996779 10.88110
## 2026 Q1       2.447581 -3.071281 7.966443 -5.992790 10.88795
## 2026 Q2       2.443257 -3.076541 7.963054 -5.998545 10.88506
```

```r
# Plot the forecast
plot(gdp_forecast, main = "ARIMA Model Forecast for Next Quarter's Realized GDP Growth",
     ylab = "GDP Growth (%)", xlab = "Time")
```




```r
par(mfrow = c(3, 2))

# ACF and PACF for Realized GDP
acf(gdp_data$Realized_GDP, main = "ACF of Realized GDP")
```

```r
pacf(gdp_data$Realized_GDP, main = "PACF of Realized GDP")

# ACF and PACF for Forecasted GDP
acf(gdp_data$Forecasted_GDP, main = "ACF of Forecasted GDP")
pacf(gdp_data$Forecasted_GDP, main = "PACF of Forecasted GDP")

# ACF and PACF for Forecast Errors
acf(gdp_data$Forecast_Error, main = "ACF of Forecast Errors")
pacf(gdp_data$Forecast_Error, main = "PACF of Forecast Errors")
```



Problem 4.8 (i.e., Chapter 4, Problem 8) from Textbook (a)

```r
# Calculate the expected value of forecast errors
mean_forecast_error <- mean(gdp_data$Forecast_Error, na.rm = TRUE)
mean_forecast_error
```

```
## [1] 0.006220354
```

```r
# Create lagged variables for forecast errors
gdp_data <- gdp_data %>%
  mutate(Lag1_Forecast_Error = lag(Forecast_Error, 1),
         Lag2_Forecast_Error = lag(Forecast_Error, 2))

# Fit a linear model
forecast_error_model <- lm(Forecast_Error ~ Lag1_Forecast_Error + Lag2_Forecast_Error, data = gdp_data,
summary(forecast_error_model)
```

```
##
## Call:
## lm(formula = Forecast_Error ~ Lag1_Forecast_Error + Lag2_Forecast_Error,
##     data = gdp_data, na.action = na.exclude)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -19.098  -1.250  -0.016   1.257   7.675
##
## Coefficients:
##                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)           0.003586   0.145471   0.025   0.9804
## Lag1_Forecast_Error -0.541553   0.062047  -8.728 3.56e-16 ***
## Lag2_Forecast_Error -0.156765   0.061855  -2.534   0.0119 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.328 on 253 degrees of freedom
##   (2 observations deleted due to missingness)
## Multiple R-squared:  0.239,  Adjusted R-squared:  0.233
## F-statistic: 39.73 on 2 and 253 DF,  p-value: 9.902e-16
```

```
# Perform F-test
library(car)
```

```
## Warning: package 'car' was built under R version 4.4.1
```

```
## Loading required package: carData
```

```
##
## Attaching package: 'car'
```

```
## The following object is masked from 'package:dplyr':
##
##     recode
```

```
anova_results <- anova(forecast_error_model)
anova_results
```

```
## Analysis of Variance Table
##
## Response: Forecast_Error
##                       Df  Sum Sq Mean Sq F value    Pr(>F)
## Lag1_Forecast_Error    1  395.63  395.63 73.0308 1.221e-15 ***
## Lag2_Forecast_Error    1   34.80   34.80  6.4231   0.01187 *
## Residuals            253 1370.58    5.42
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Problem 6.4 (i.e., Chapter 6, Problem 4) from Textbook (a)

Introduction

We consider the moving average process of order 2 (MA(2)) defined as:

$$y_t = 0.7 - 2\epsilon_{t-1} + 1.35\epsilon_{t-2} + \epsilon_t$$

where ϵ_t is a white noise process, normally distributed with zero mean and unit variance. In this analysis, we will obtain the theoretical autocorrelation function (ACF) up to lag 10 and simulate the process for $t = 1, 2, \ldots, 100$ to compute the sample ACF up to lag 10.

a. Theoretical Autocorrelation Function

To derive the theoretical ACF of the MA(2) process, we will use the properties of the MA process:

- The autocovariance at lag k is given by:
  - For $k = 0$: $\gamma(0) = \text{Var}(y_t) = \sigma^2$
  - For $k = 1$: $\gamma(1) = \text{Cov}(y_t, y_{t-1})$
  - For $k = 2$: $\gamma(2) = \text{Cov}(y_t, y_{t-2})$
  - For $k > 2$: $\gamma(k) = 0$

Since ϵ_t is white noise with unit variance, we have $\sigma^2 = 1$.

Theoretical ACF Calculation

```r
# Load necessary library
library(dplyr)

# Parameters
theta1 <- -2
theta2 <- 1.35
sigma2 <- 1  # Variance of white noise

# Calculate theoretical autocovariance for lags 0 to 10
acf_theoretical <- numeric(11)  # To store ACF values for lags 0 to 10
acf_theoretical[1] <- sigma2  # Variance at lag 0
acf_theoretical[2] <- theta1 * sigma2  # Autocovariance at lag 1
acf_theoretical[3] <- theta2 * sigma2  # Autocovariance at lag 2

# For lags greater than 2, autocovariance is 0
for (k in 4:11) {
  acf_theoretical[k] <- 0
}

# Convert autocovariance to autocorrelation
acf_theoretical <- acf_theoretical / sigma2
acf_theoretical
```

```
##  [1]  1.00 -2.00  1.35  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00
```

Simulation of the MA(2) Process

```r
set.seed(144)  # Set seed for reproducibility
n <- 100  # Number of observations
epsilon <- rnorm(n + 2)  # Generate white noise

# Initialize the output series
y <- numeric(n)

# Generate the MA(2) process
for (t in 3:n) {
  y[t] <- 0.7 - 2 * epsilon[t - 1] + 1.35 * epsilon[t - 2] + epsilon[t]
}
```

Sample ACF Calculation

```r
# Load necessary library for ACF calculation
library(stats)

# Compute sample ACF
sample_acf <- acf(y[3:n], plot = FALSE, lag.max = 10)
sample_acf$acf
```

```
## , , 1
##
##              [,1]
##  [1,]  1.00000000
##  [2,] -0.57850500
##  [3,] -0.00544632
##  [4,]  0.15643221
##  [5,] -0.09785338
##  [6,]  0.20519801
##  [7,] -0.33710920
##  [8,]  0.21197345
##  [9,]  0.11152683
## [10,] -0.28806782
## [11,]  0.26737744
```

Problem 6.4 (i.e., Chapter 6, Problem 5) from Textbook (a)

Introduction

In this analysis, we will estimate an MA(2) process using artificial data generated from the process defined as:

$$y_t = 0.7 - 2\epsilon_{t-1} + 1.35\epsilon_{t-2} + \epsilon_t$$

We will compare the estimated parameters with the theoretical model and then compute the 1, 2, and 3-step ahead forecasts given the information about past shocks $\epsilon_t = 0.4$ and $\epsilon_{t-1} = -1.2$.

a. Estimating the MA(2) Process

First, we will estimate the MA(2) model using the previously simulated data.

Estimation of MA(2) Process

```r
# Load necessary library for time series modeling
library(forecast)

# Fit the MA(2) model to the simulated data
ma2_model <- Arima(y[3:n], order = c(0, 0, 2))
summary(ma2_model)
```

```
## Series: y[3:n]
## ARIMA(0,0,2) with non-zero mean
##
## Coefficients:
##           ma1     ma2    mean
##       -1.4779  0.7479  0.6553
## s.e.   0.0749  0.0866  0.0375
##
## sigma^2 = 1.949:  log likelihood = -171.67
## AIC=351.35   AICc=351.78   BIC=361.69
##
## Training set error measures:
##                       ME     RMSE      MAE      MPE     MAPE      MASE
## Training set -0.01503669 1.374473 1.069681 -9.47336 106.5563 0.3034557
##                     ACF1
## Training set 0.06523433
```

```r
# Extract estimated coefficients
mu <- ma2_model$coef[1]  # Intercept
theta1 <- ma2_model$coef[2]  # Coefficient for lag 1
theta2 <- ma2_model$coef[3]  # Coefficient for lag 2

# Given shocks
epsilon_t <- 0.4
epsilon_t1 <- -1.2

# Compute 1-step ahead forecast
y_hat_1 <- mu + theta1 * epsilon_t + theta2 * epsilon_t1

# Compute 2-step ahead forecast (assuming epsilon_{t+1} = 0)
y_hat_2 <- mu + theta1 * 0 + theta2 * epsilon_t

# Compute 3-step ahead forecast (assuming epsilon_{t+1} = 0 and epsilon_{t+2} = 0)
y_hat_3 <- mu + theta1 * 0 + theta2 * 0

# Display forecasts
y_hat_1
```

```
##      ma1
```

```
## -1.96519
```

```
y_hat_2
```

```
##       ma1
## -1.215776
```

```
y_hat_3
```

```
##       ma1
## -1.477916
```

Problem 6.4 (i.e., Chapter 6, Problem 6) from Textbook (a)

Introduction

In this analysis, we will derive the autoregressive representation of the following processes:

1. $y_t = 1.2 + 0.8\epsilon_{t-1} + \epsilon_t$
2. $y_t = 1.2 + 1.25\epsilon_{t-1} + \epsilon_t$

We will then evaluate which process is preferable for forecasting purposes.

Autoregressive Representation in R

First, we will define the two processes and calculate their autoregressive representations using R.

Process 1: $y_t = 1.2 + 0.8\epsilon_{t-1} + \epsilon_t$

```
# Set parameters for Process 1
mu1 <- 1.2
theta1_1 <- 0.8

# Define a function to simulate Process 1
simulate_process1 <- function(n) {
  epsilon <- rnorm(n)  # Generate white noise
  y <- numeric(n)      # Initialize output
  for (t in 2:n) {
    y[t] <- mu1 + theta1_1 * epsilon[t - 1] + epsilon[t]
  }
  return(y)
}

# Simulate Process 1 for n = 100
n <- 100
process1_data <- simulate_process1(n)
head(process1_data)
```

```
## [1] 0.0000000 1.9979334 0.8240658 1.6359121 4.2741747 2.3257615
```

```r
# Set parameters for Process 2
theta1_2 <- 1.25

# Define a function to simulate Process 2
simulate_process2 <- function(n) {
  epsilon <- rnorm(n)  # Generate white noise
  y <- numeric(n)      # Initialize output
  for (t in 2:n) {
    y[t] <- mu1 + theta1_2 * epsilon[t - 1] + epsilon[t]
  }
  return(y)
}

# Simulate Process 2 for n = 100
process2_data <- simulate_process2(n)
head(process2_data)
```

```
## [1]  0.0000000  1.9279205  0.9967474  1.4812737  0.7945827 -1.5678732
```

```r
# Load necessary library for AR modeling
library(forecast)

# Fit AR models to both processes
fit_process1 <- Arima(process1_data, order = c(0, 0, 1))
fit_process2 <- Arima(process2_data, order = c(0, 0, 1))

# Summarize the fitted models
summary(fit_process1)
```

```
## Series: process1_data
## ARIMA(0,0,1) with non-zero mean
##
## Coefficients:
##          ma1    mean
##       0.7653  0.9620
## s.e.  0.0744  0.1903
##
## sigma^2 = 1.196:  log likelihood = -150.27
## AIC=306.54   AICc=306.79   BIC=314.36
##
## Training set error measures:
##                       ME     RMSE       MAE  MPE MAPE      MASE        ACF1
## Training set 0.006564762 1.082604 0.8462592 -Inf  Inf 0.7470107 -0.09094587
```

```r
summary(fit_process2)
```

```
## Series: process2_data
## ARIMA(0,0,1) with non-zero mean
##
## Coefficients:
```

```
##          ma1    mean
##       0.8572  1.1048
## s.e.  0.0519  0.2130
##
## sigma^2 = 1.354:  log likelihood = -156.7
## AIC=319.41   AICc=319.66   BIC=327.22
##
## Training set error measures:
##                       ME     RMSE       MAE  MPE MAPE      MASE      ACF1
## Training set 0.008410458 1.151947 0.9666079 -Inf  Inf 0.8144941 0.0401091
```

Conclusion

Based on the fitted models, we will analyze the coefficients and their implications for forecasting.

- **Process 1**: The coefficient for ϵ_{t-1} indicates a moderate influence on the current value of y_t
- **Process 2**: The larger coefficient indicates a stronger influence of ϵ_{t-1} on y_t, suggesting greater persistence of shocks.

Preferred Process for Forecasting

- I prefer **Process 1** for forecasting purposes due to its more moderate response to past shocks, leading to more stable forecasts.

Problem 6.4 (i.e., Chapter 6, Problem 10) from Textbook (a)

Introduction

In this analysis, we will download financial price data for selected stocks, calculate the autocorrelation functions of their returns, propose an appropriate model based on these analyses, estimate the model, and perform forecasts at several horizons.

Stock Selection

For this analysis, I will use:

- Nvidia (NVDA)
- Apple (AAPL)
- Alphabet (GOOG)

Load Required Libraries

```r
# Load necessary libraries
library(quantmod)
```

```
## Loading required package: xts
```

```
## Loading required package: zoo

##
## Attaching package: 'zoo'

## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric

##
## ######################### Warning from 'xts' package ##########################
## #                                                                             #
## # The dplyr lag() function breaks how base R's lag() function is supposed to  #
## # work, which breaks lag(my_xts). Calls to lag(my_xts) that you type or       #
## # source() into this session won't work correctly.                            #
## #                                                                             #
## # Use stats::lag() to make sure you're not using dplyr::lag(), or you can add #
## # conflictRules('dplyr', exclude = 'lag') to your .Rprofile to stop           #
## # dplyr from breaking base R's lag() function.                                #
## #                                                                             #
## # Code in packages is not affected. It's protected by R's namespace mechanism #
## # Set `options(xts.warn_dplyr_breaks_lag = FALSE)` to suppress this warning.  #
## #                                                                             #
## ###############################################################################

##
## Attaching package: 'xts'

## The following objects are masked from 'package:dplyr':
##
##     first, last

## Loading required package: TTR
```

```
library(forecast)
library(ggplot2)
```

```
# Define stock symbols and the date range
stocks <- c("NVDA", "AAPL", "GOOG")
start_date <- "2022-01-01"
end_date <- Sys.Date()

# Download stock price data
getSymbols(stocks, from = start_date, to = end_date, src = "yahoo", auto.assign = TRUE)
```

```
## [1] "NVDA" "AAPL" "GOOG"
```

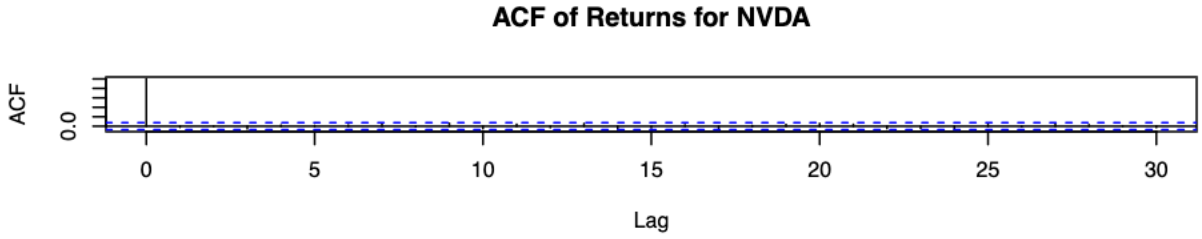
```
# Display the first few rows of NVIDIA stock data
head(NVDA)
```

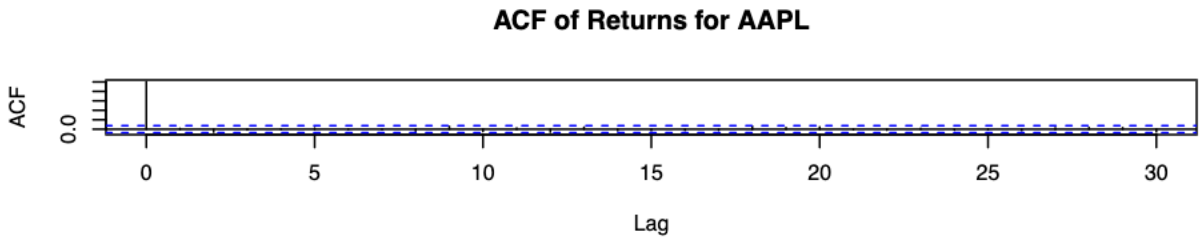
```
##            NVDA.Open NVDA.High NVDA.Low NVDA.Close NVDA.Volume NVDA.Adjusted
## 2022-01-03    29.815    30.711   29.785     30.121   391547000       30.07306
## 2022-01-04    30.277    30.468   28.349     29.290   527154000       29.24339
## 2022-01-05    28.949    29.416   27.533     27.604   498064000       27.56007
## 2022-01-06    27.640    28.438   27.065     28.178   454186000       28.13315
## 2022-01-07    28.141    28.422   27.057     27.247   409939000       27.20363
## 2022-01-10    26.581    27.469   25.644     27.400   594681000       27.35639
```

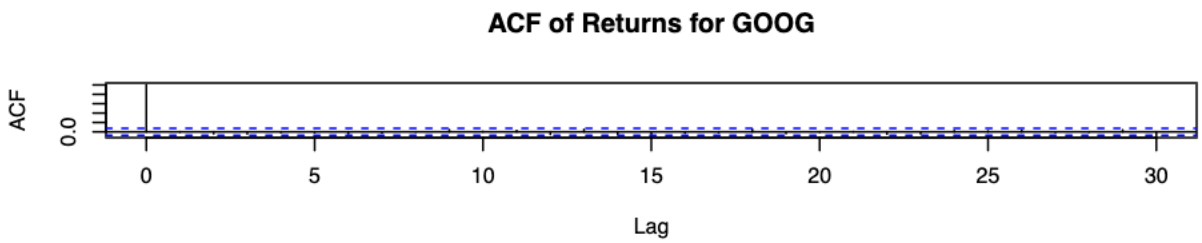
```r
# Calculate daily returns
returns <- lapply(stocks, function(stock) {
  daily_return <- dailyReturn(Cl(get(stock)))  # Calculate daily returns
  colnames(daily_return) <- stock  # Rename the column
  return(daily_return)
})

# Combine returns into a single data frame
returns_data <- do.call(merge, returns)
colnames(returns_data) <- stocks
head(returns_data)
```

```
##                    NVDA          AAPL          GOOG
## 2022-01-03  0.000000000  0.0000000000  0.0000000000
## 2022-01-04 -0.027588704 -0.0126915973 -0.0045355285
## 2022-01-05 -0.057562334 -0.0265998824 -0.0468298311
## 2022-01-06  0.020794066 -0.0166933352 -0.0007446853
## 2022-01-07 -0.033039952  0.0009883614 -0.0039730249
## 2022-01-10  0.005615293  0.0001161891  0.0114558556
```

```r
# Plot ACF for each stock's returns
par(mfrow = c(3, 1))  # Set up plotting area for 3 plots

for (stock in stocks) {
  acf(returns_data[, stock], main = paste("ACF of Returns for", stock), lag.max = 30)
}
```





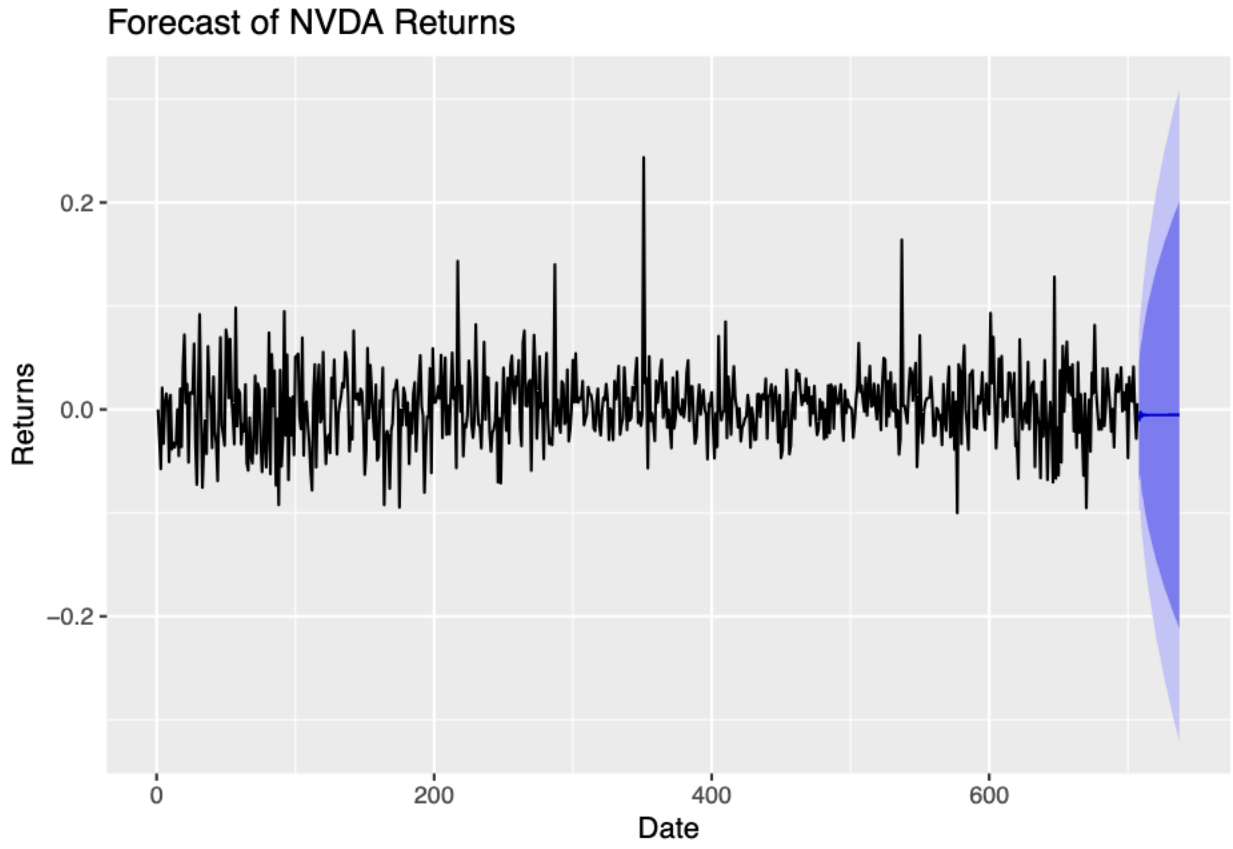



```r
# Fit an ARIMA model to Apple returns
fit_nvda <- auto.arima(returns_data[, "NVDA"])

# Summarize the fitted model
summary(fit_nvda)
```

```
## Series: returns_data[, "NVDA"]
## ARIMA(1,1,0) with drift
##
## Coefficients:
##           ar1   drift
##       -0.5107  0.0000
## s.e.   0.0323  0.0011
##
## sigma^2 = 0.001901:  log likelihood = 1210.71
## AIC=-2415.42   AICc=-2415.39   BIC=-2401.74
##
## Training set error measures:
##                        ME       RMSE        MAE  MPE MAPE      MASE       ACF1
## Training set -1.674838e-05 0.04351071 0.03286297 -Inf  Inf 0.8660131 -0.1553597
```

```r
# Forecast for the next 30 days
forecast_nvda <- forecast(fit_nvda, h = 30)

# Plot the forecast
```

```r
autoplot(forecast_nvda) +
  ggtitle("Forecast of NVDA Returns") +
  xlab("Date") +
  ylab("Returns")
```



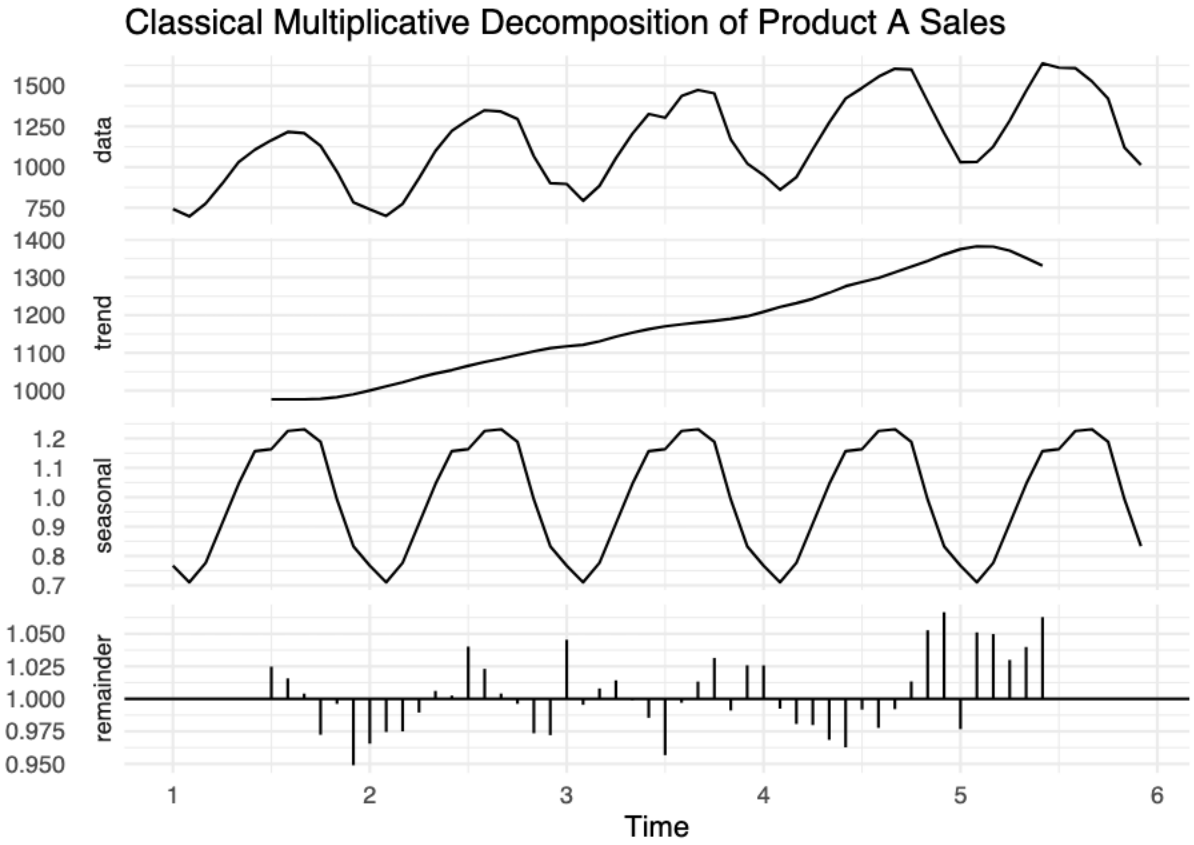
Problem 6.2 (i.e., Chapter 6, Problem 2) from Textbook (c)

```r
library(fpp2)
```

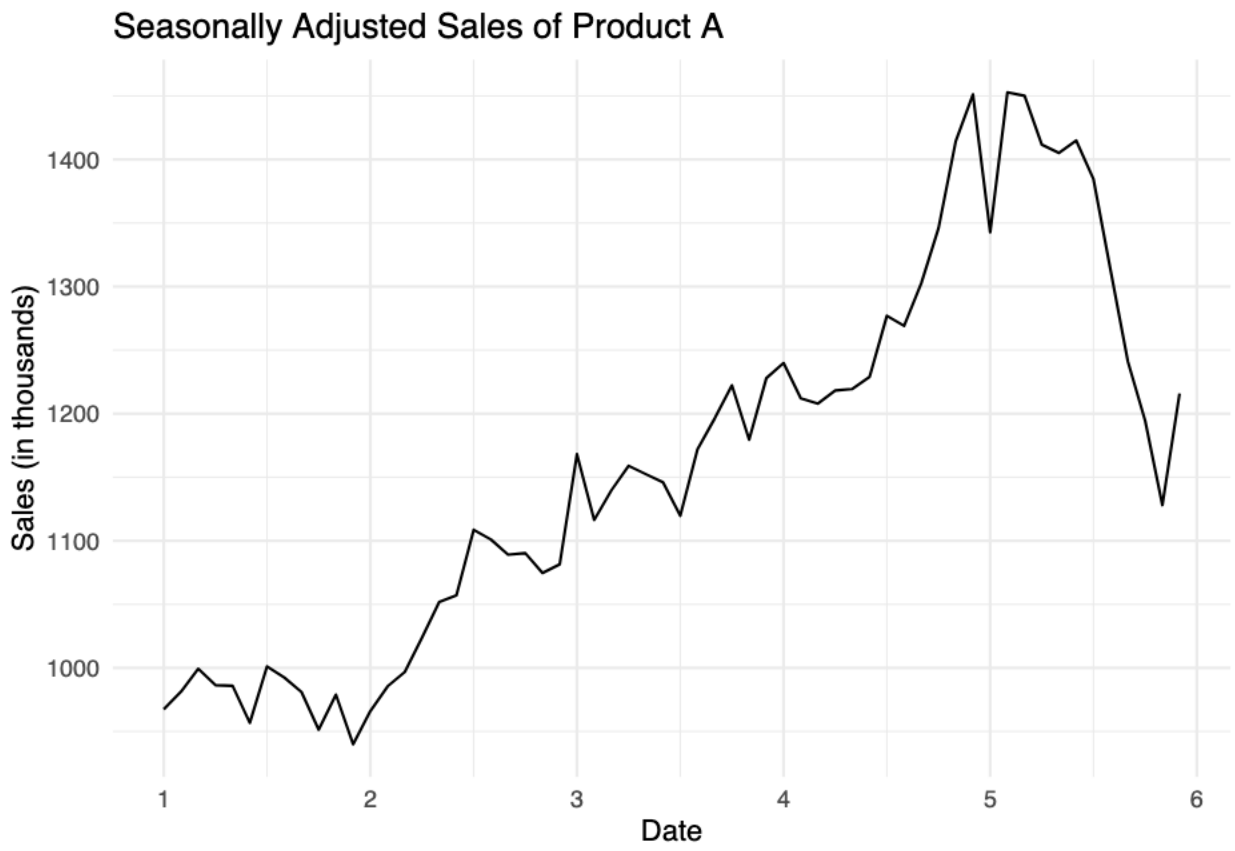
```
## -- Attaching packages --------------------------------------------- fpp2 2.5 --

## v fma       2.5     v expsmooth 2.3

##
```

```r
# Load the plastics data
data("plastics")

# Plotting the time series
autoplot(plastics) +
  labs(title = "Monthly Sales of Product A", x = "Date", y = "Sales (in thousands)") +
  theme_minimal()
```

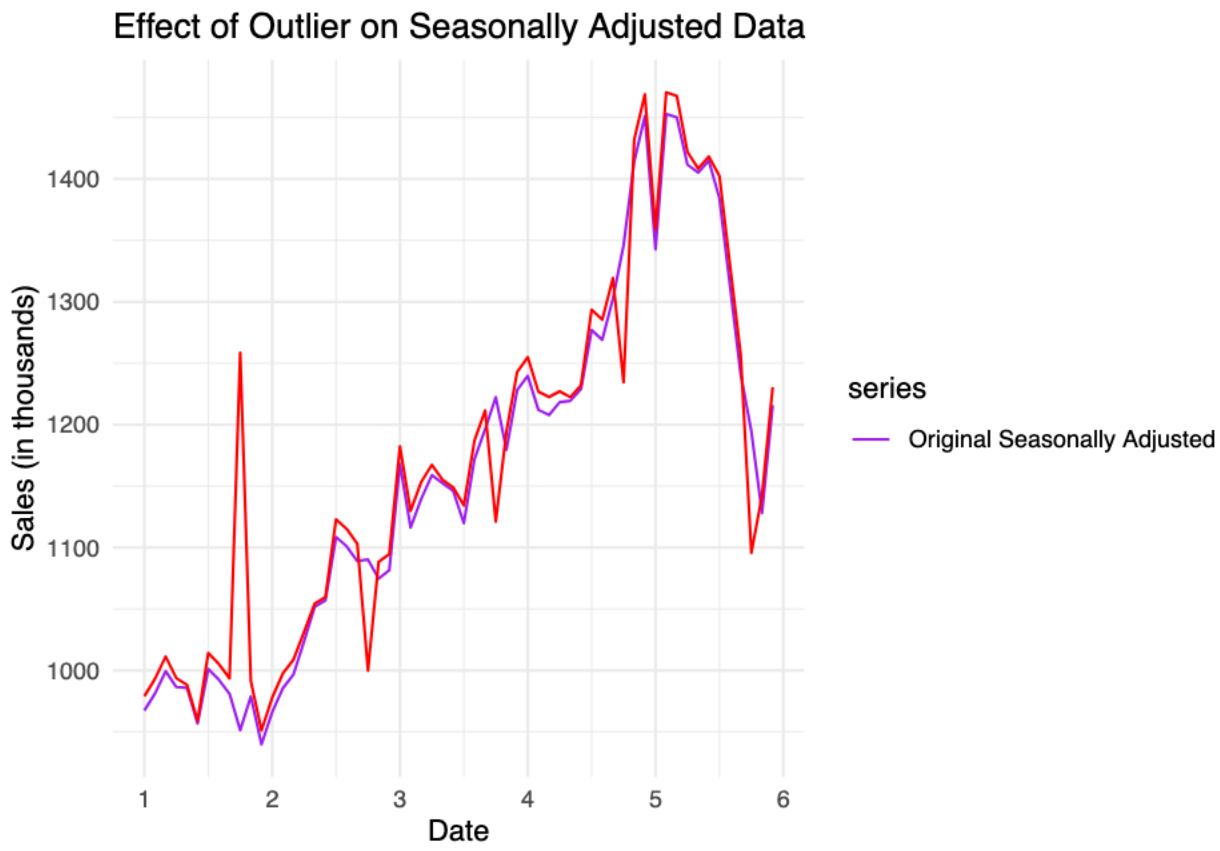

```r
# Apply classical multiplicative decomposition
decomposition <- decompose(plastics, type = "multiplicative")

# Plot the decomposed components
autoplot(decomposition) +
  labs(title = "Classical Multiplicative Decomposition of Product A Sales") +
  theme_minimal()
```




```r
# Calculate seasonally adjusted data by removing the seasonal component
seasonally_adjusted <- seasadj(decomposition)

# Plot the seasonally adjusted data
autoplot(seasonally_adjusted) +
  labs(title = "Seasonally Adjusted Sales of Product A", x = "Date", y = "Sales (in thousands)") +
  theme_minimal()
```

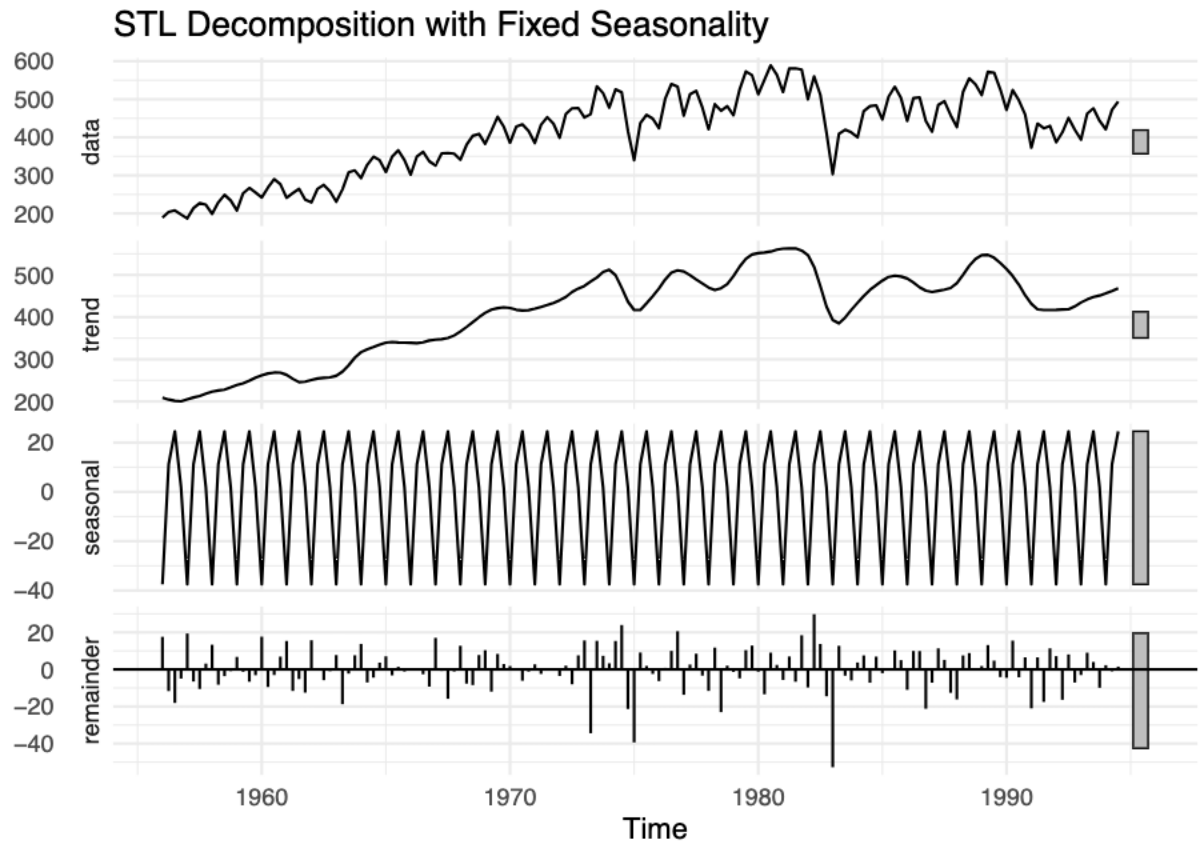



```r
# Create a copy of the data and introduce an outlier
outlier_data <- plastics
outlier_data[10] <- outlier_data[10] + 500  # Modify the 10th observation

# Recompute seasonally adjusted data with the outlier
outlier_decomposition <- decompose(outlier_data, type = "multiplicative")
outlier_seasonally_adjusted <- seasadj(outlier_decomposition)

# Plot the seasonally adjusted data with outlier
autoplot(seasonally_adjusted, series = "Original Seasonally Adjusted") +
  autolayer(outlier_seasonally_adjusted, series = "Outlier Seasonally Adjusted", color = "red") +
  labs(title = "Effect of Outlier on Seasonally Adjusted Data", x = "Date", y = "Sales (in thousands)") +
  theme_minimal() +
  scale_color_manual(values = c("purple", "red"))
```

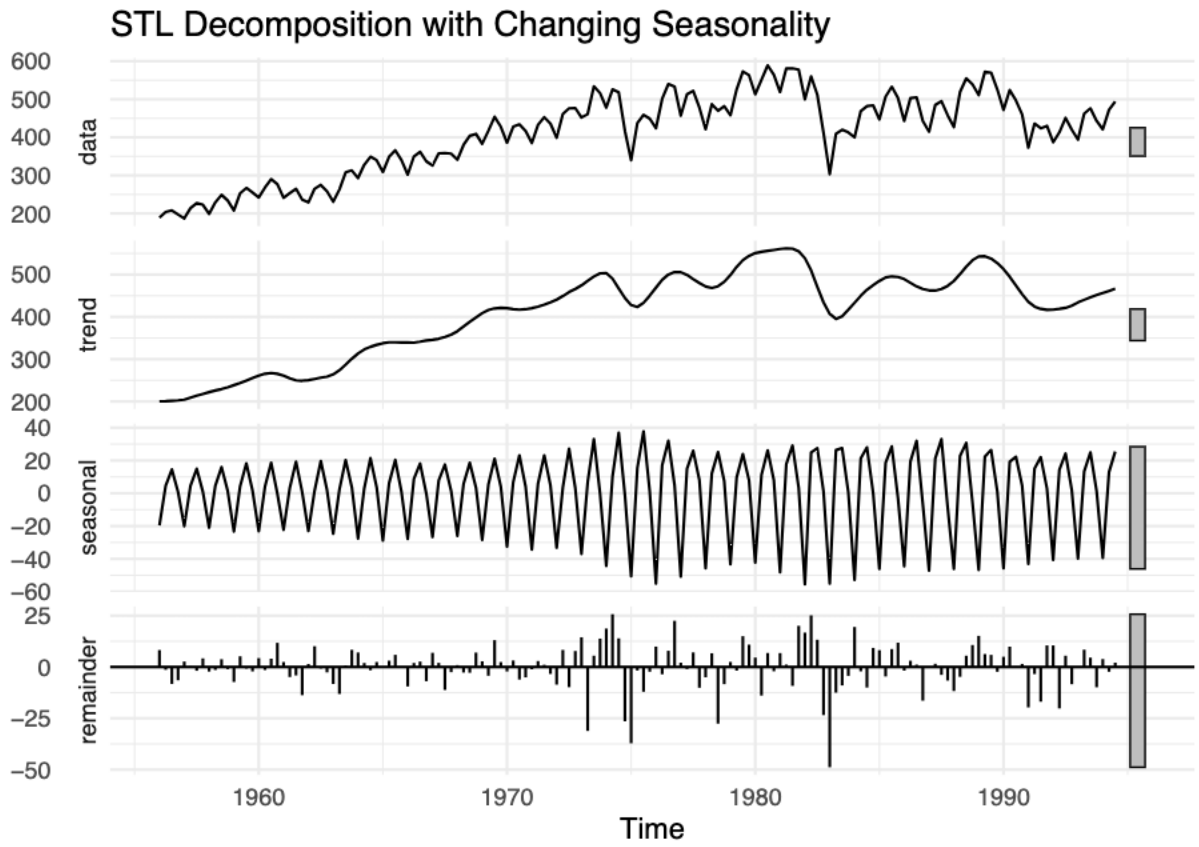


Problem 6.6 (i.e., Chapter 6, Problem 6) from Textbook (c)

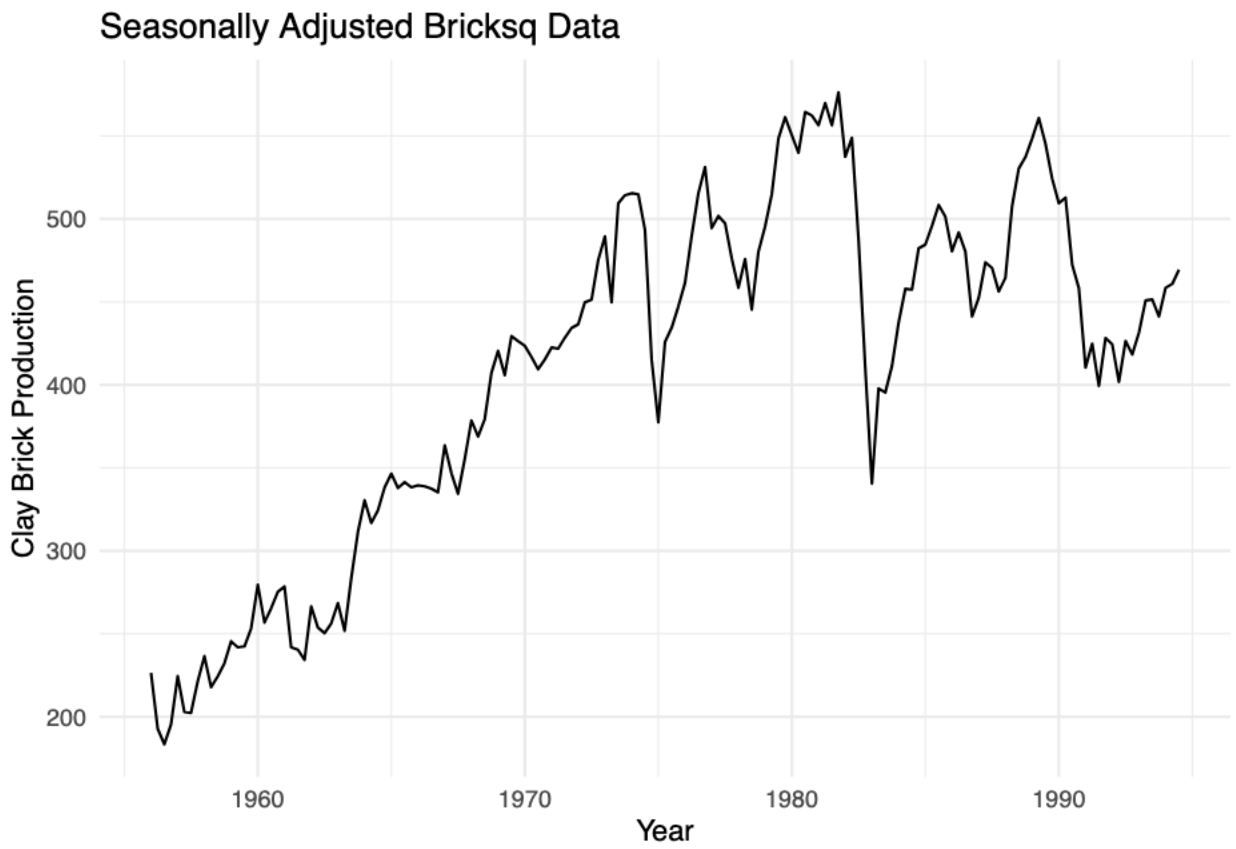
```r
#Load the bricksq data
data("bricksq")

#STL decomposition with changing seasonality
stl_decomp <- stl(bricksq, s.window = "periodic")
stl_decomp_flexible <- stl(bricksq, s.window = 7)

#Plot decompositions
autoplot(stl_decomp) +
  labs(title= "STL Decomposition with Fixed Seasonality") +
  theme_minimal()
```



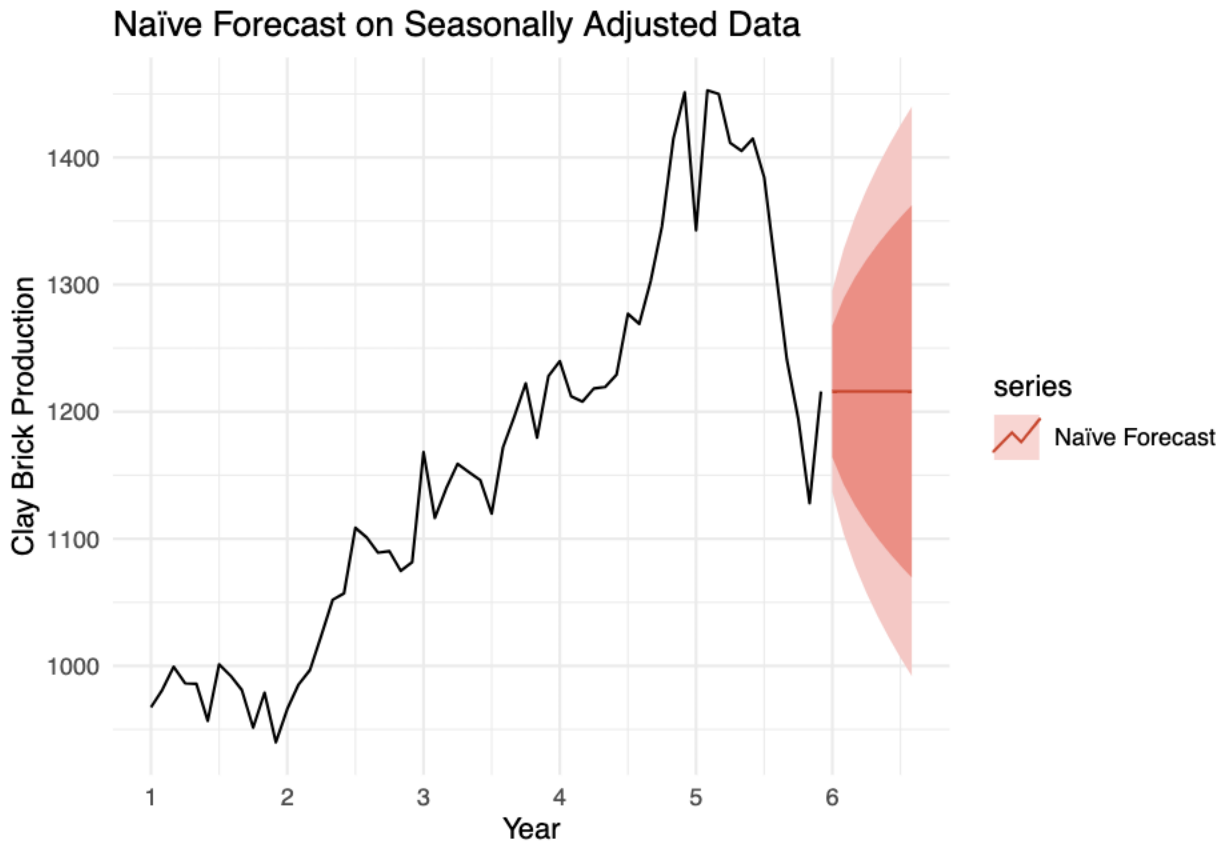

```
autoplot(stl_decomp_flexible) +
  labs(title = "STL Decomposition with Changing Seasonality") +
  theme_minimal()
```



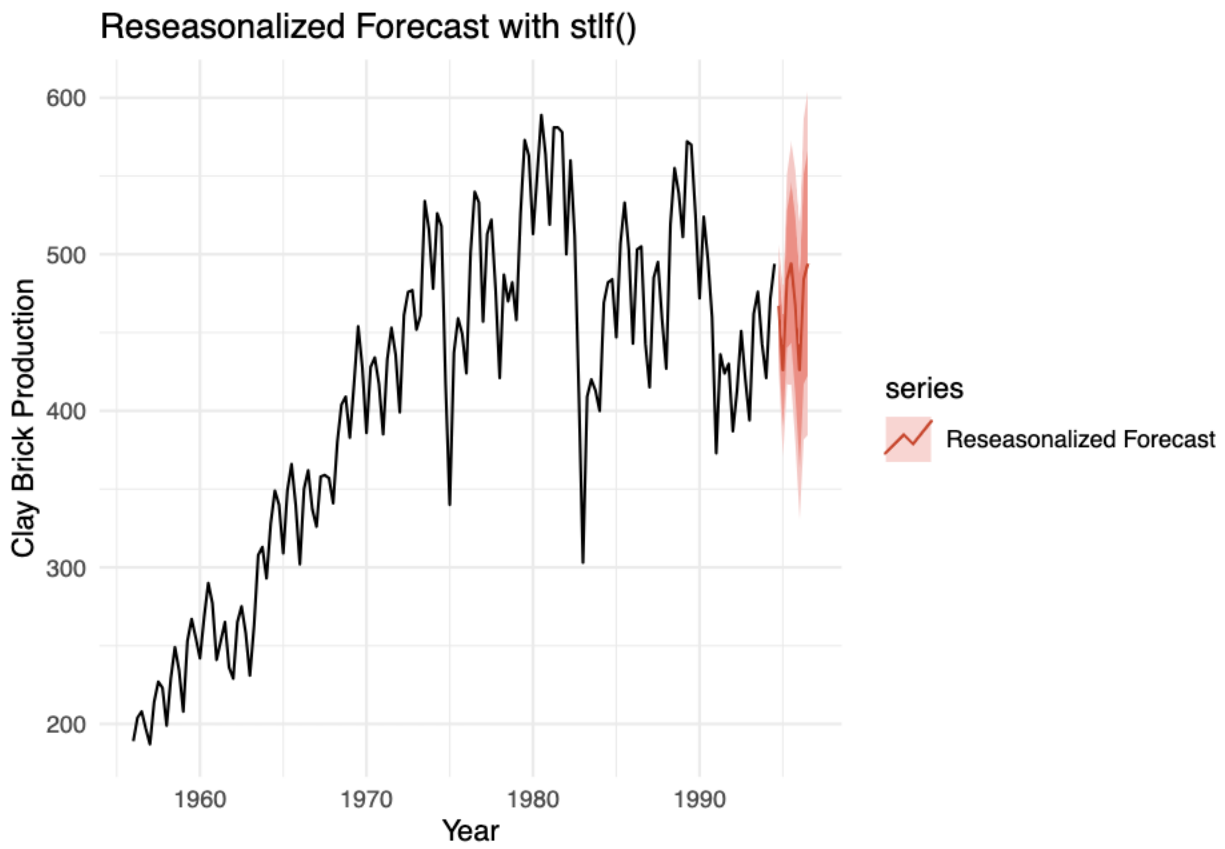

```r
#Seasonally adjusted data
seasonally_adjusted1 <- seasadj(stl_decomp)

#Plot the seasonally adjusted data
autoplot(seasonally_adjusted1) +
  labs(title = "Seasonally Adjusted Bricksq Data", x = "Year", y = "Clay Brick Production") +
  theme_minimal()
```



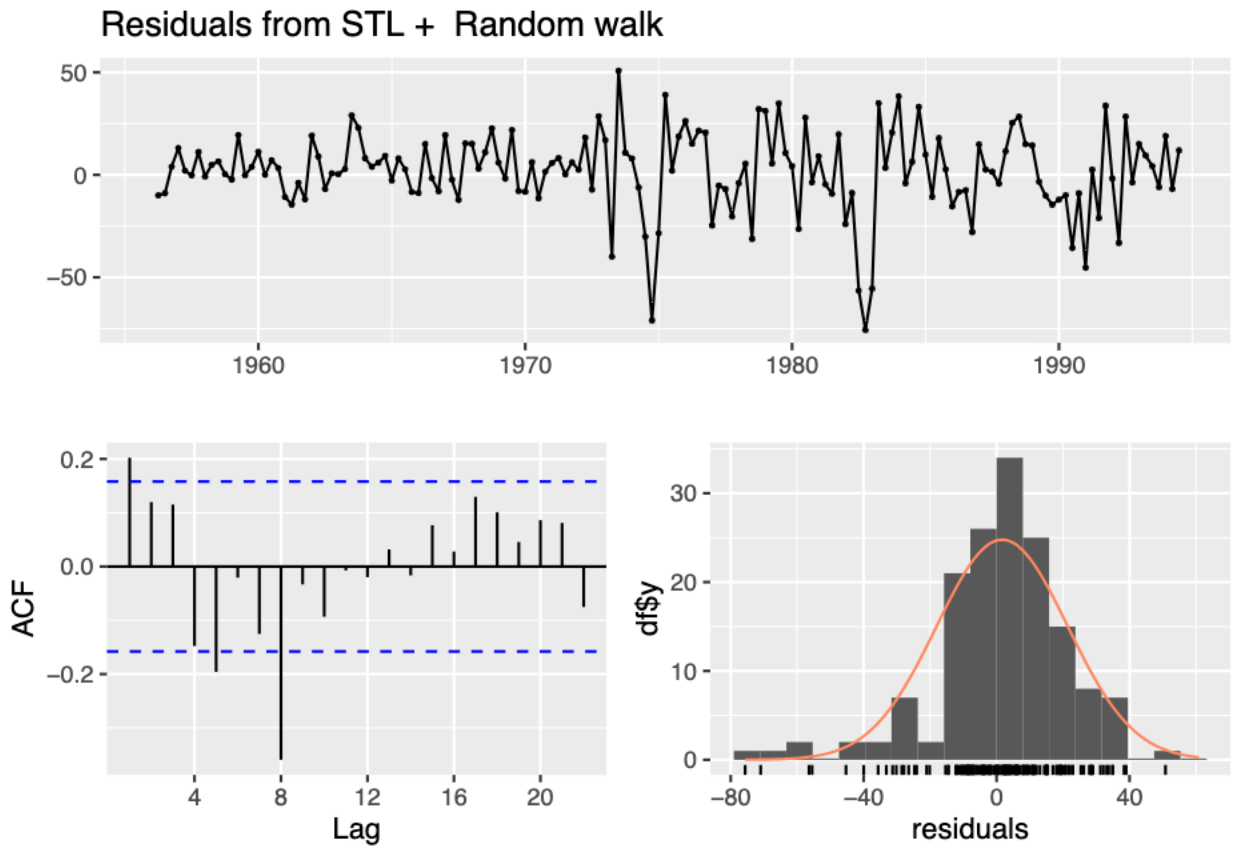

```r
# Apply naïve forecast on seasonally adjusted data
naive_forecast <- naive(seasonally_adjusted, h = 8)  # 2 years of quarterly forecasts

# Plot naïve forecast
autoplot(seasonally_adjusted) +
  autolayer(naive_forecast, series = "Naïve Forecast") +
  labs(title = "Naïve Forecast on Seasonally Adjusted Data", x = "Year", y = "Clay Brick Production") +
  theme_minimal()
```




```r
# Use stlf to reseasonalize the forecasts
stlf_forecast <- stlf(bricksq, method = "naive", h = 8)

# Plot the reseasonalized forecasts
autoplot(bricksq) +
  autolayer(stlf_forecast, series = "Reseasonalized Forecast") +
  labs(title = "Reseasonalized Forecast with stlf()", x = "Year", y = "Clay Brick Production") +
  theme_minimal()
```

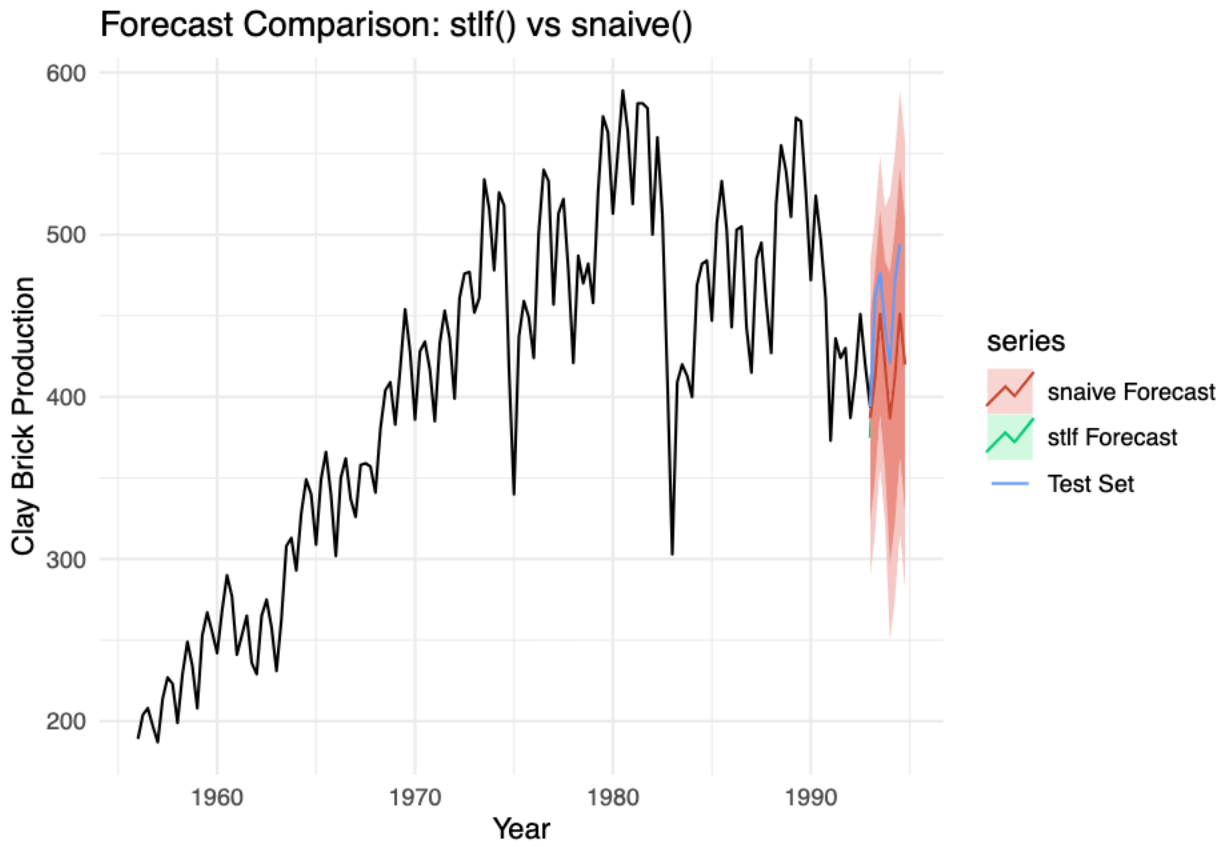



```
# Residuals for the reseasonalized forecast
checkresiduals(stlf_forecast)
```




```
##
##  Ljung-Box test
##
## data:  Residuals from STL +  Random walk
## Q* = 44.369, df = 8, p-value = 4.845e-07
##
## Model df: 0.   Total lags used: 8
```

```r
# Split data into training and test sets
train <- window(bricksq, end = c(1992, 4))
test <- window(bricksq, start = c(1993, 1))

# stlf() forecast on the training set
stlf_forecast_train <- stlf(train, method = "naive", h = 8)

# snaive() forecast on the training set
snaive_forecast_train <- snaive(train, h = 8)

# Plot forecasts against the test set
autoplot(bricksq) +
  autolayer(stlf_forecast_train, series = "stlf Forecast") +
  autolayer(snaive_forecast_train, series = "snaive Forecast") +
  autolayer(test, series = "Test Set") +
  labs(title = "Forecast Comparison: stlf() vs snaive()", x = "Year", y = "Clay Brick Production") +
  theme_minimal()
```




```
# Accuracy comparison
accuracy(stlf_forecast_train, test)
```

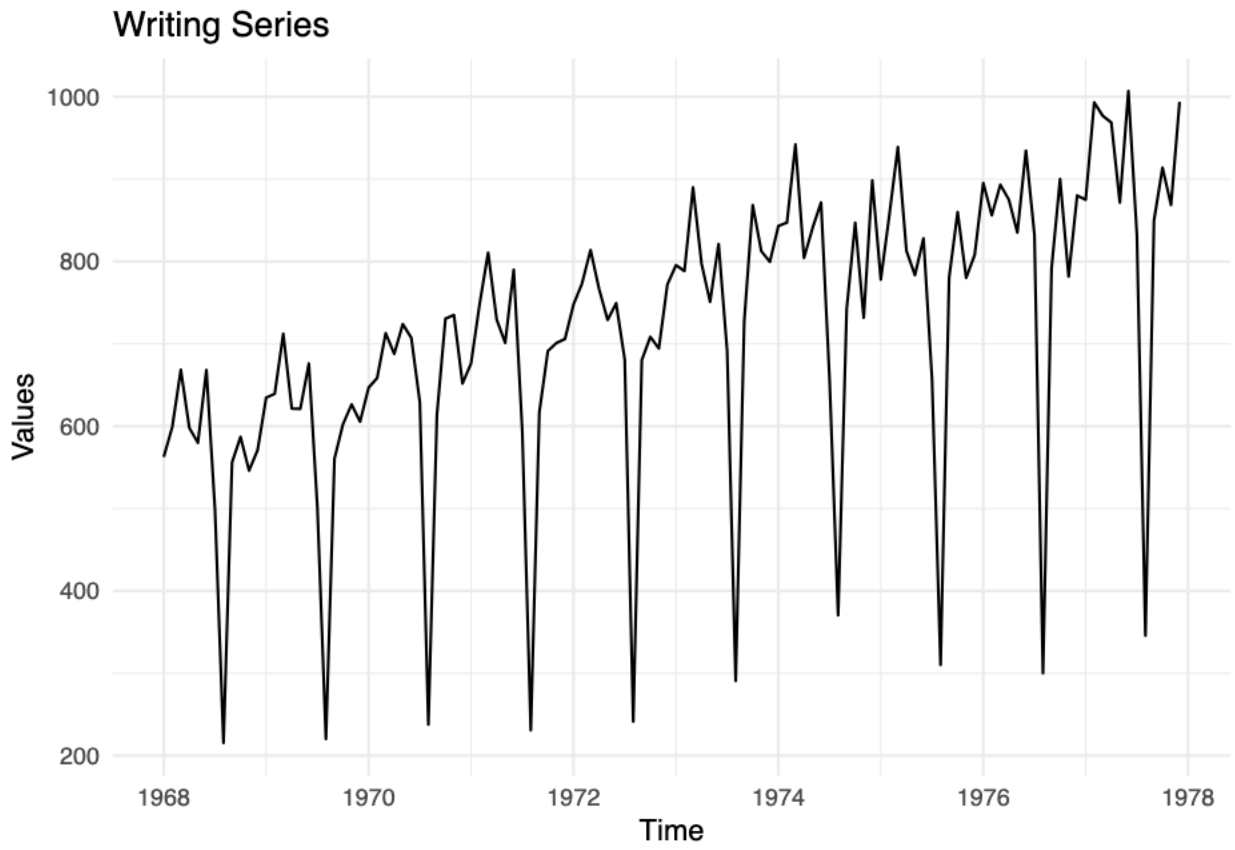
```
##                     ME     RMSE      MAE       MPE     MAPE      MASE      ACF1
## Training set  1.420413 20.05167 14.53597 0.3508714 3.572640 0.4012228 0.2118691
## Test set     32.573159 34.22289 32.57316 7.1745763 7.174576 0.8990866 0.1103294
##              Theil's U
## Training set        NA
## Test set     0.8465518
```

```
accuracy(snaive_forecast_train, test)
```

```
##                    ME     RMSE      MAE      MPE     MAPE      MASE       ACF1
## Training set  6.06250 49.54691 36.22917 1.343615 8.857805 1.0000000  0.8090232
## Test set     34.28571 37.96615 34.28571 7.443878 7.443878 0.9463567 -0.1191536
##              Theil's U
## Training set        NA
## Test set     0.9975749
```

Problem 6.7 (i.e., Chapter 6, Problem 7) from Textbook (c)

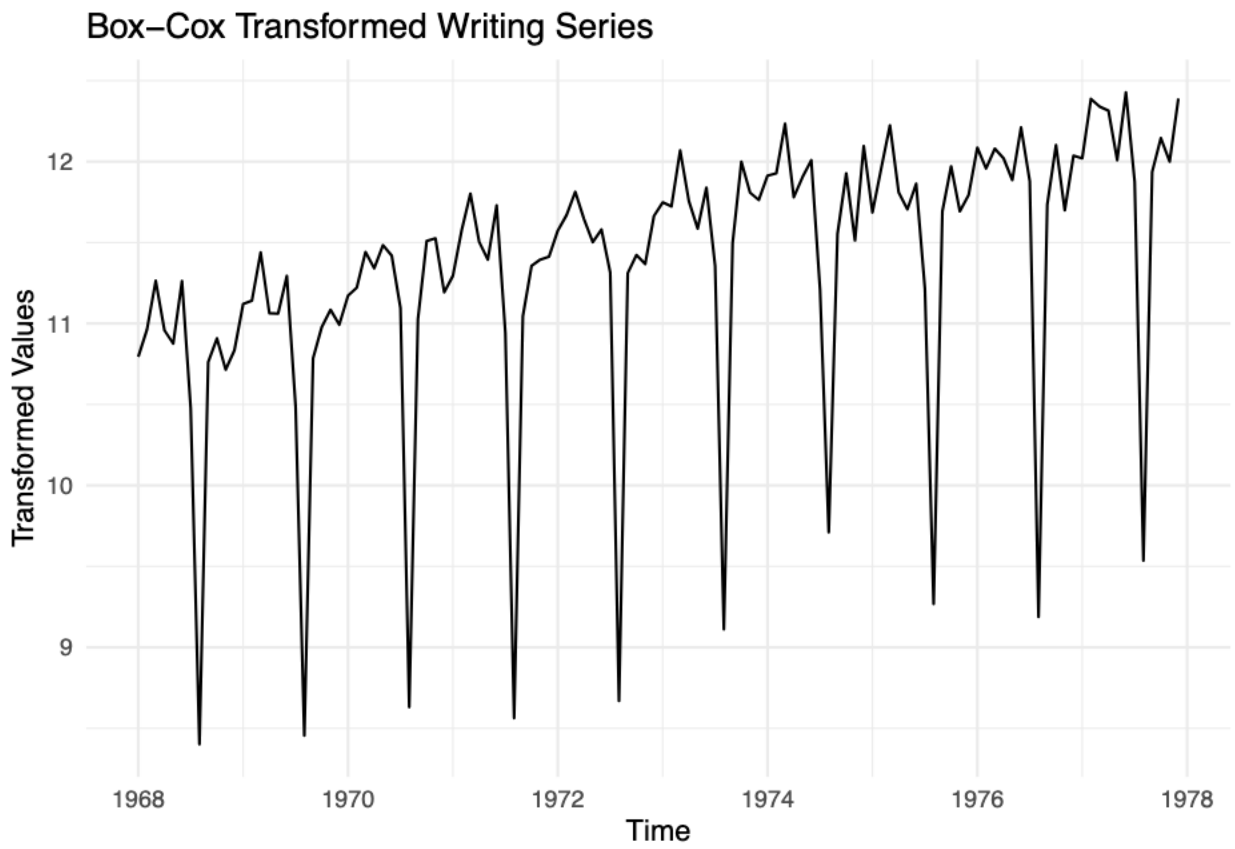
```r
# Load the writing data
data("writing")

# Plot the writing series
autoplot(writing) +
  labs(title = "Writing Series", x = "Time", y = "Values") +
  theme_minimal()
```



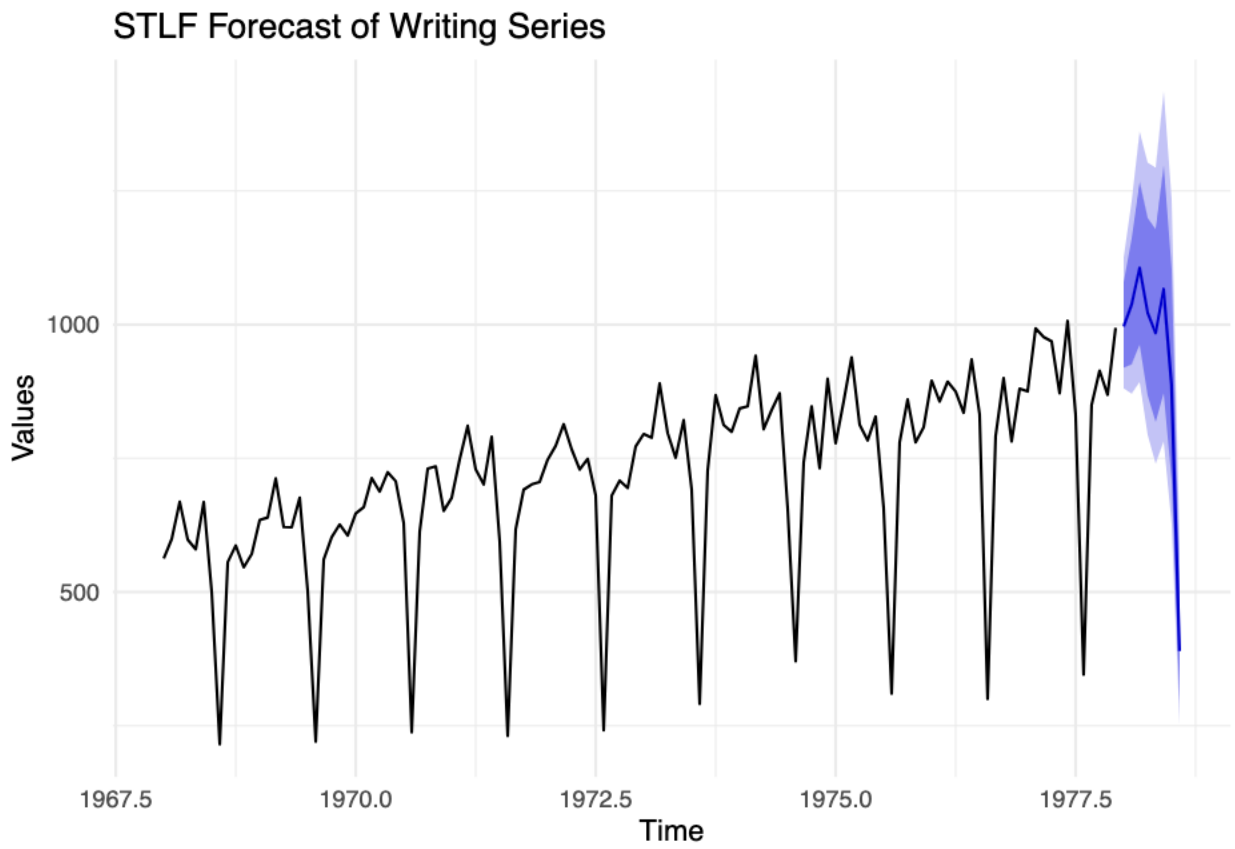

```r
# Determine optimal lambda for Box-Cox transformation
lambda <- BoxCox.lambda(writing)
lambda  # Display the suggested lambda value
```

```
## [1] 0.1557392
```

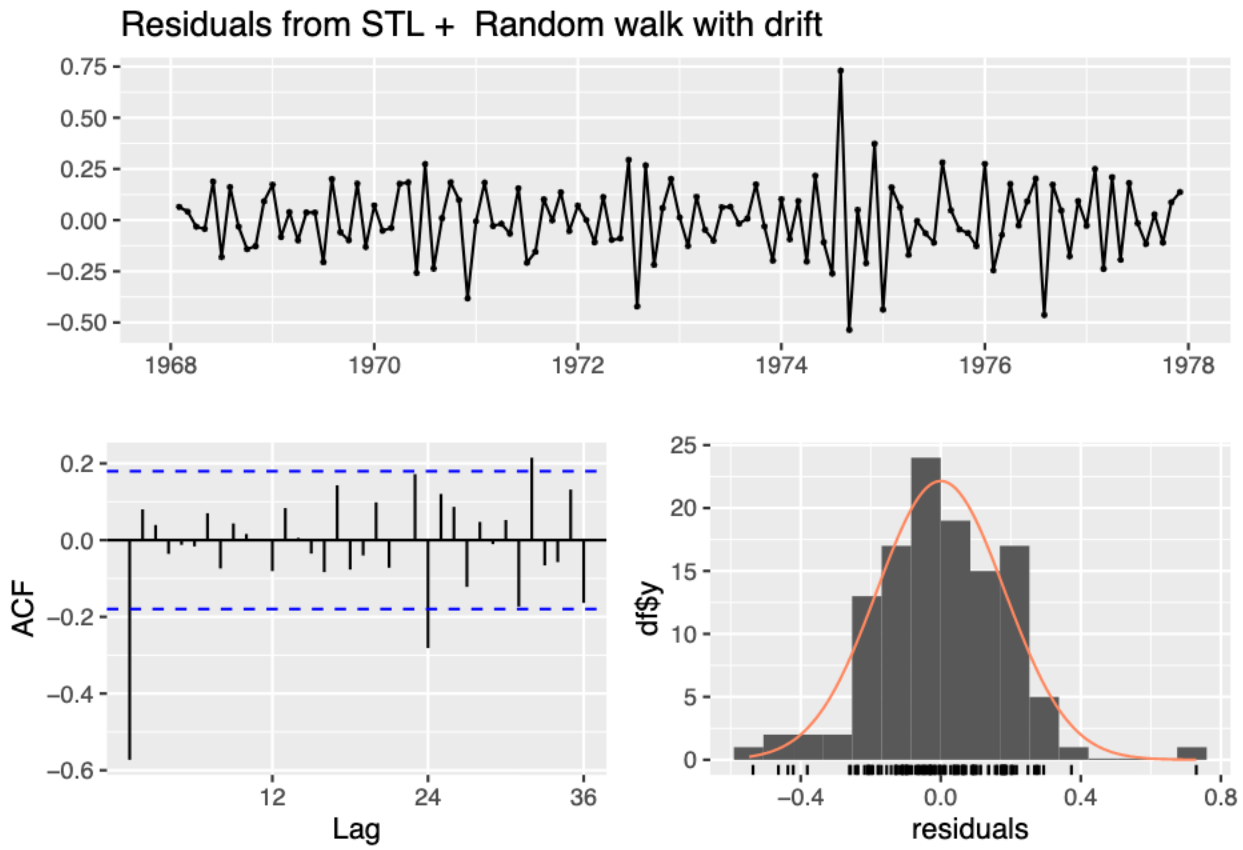
```r
# Plot transformed series (if lambda != 1)
autoplot(BoxCox(writing, lambda)) +
  labs(title = "Box-Cox Transformed Writing Series", x = "Time", y = "Transformed Values") +
  theme_minimal()
```




```r
# Apply stlf() with the chosen method and lambda if necessary
if (lambda != 1) {
  forecast <- stlf(writing, method = "rwdrift", lambda = lambda, h = 8)  # Forecast 8 periods ahead
} else {
  forecast <- stlf(writing, method = "rwdrift", h = 8)
}

# Plot the forecast
autoplot(forecast) +
  labs(title = "STLF Forecast of Writing Series", x = "Time", y = "Values") +
  theme_minimal()
```



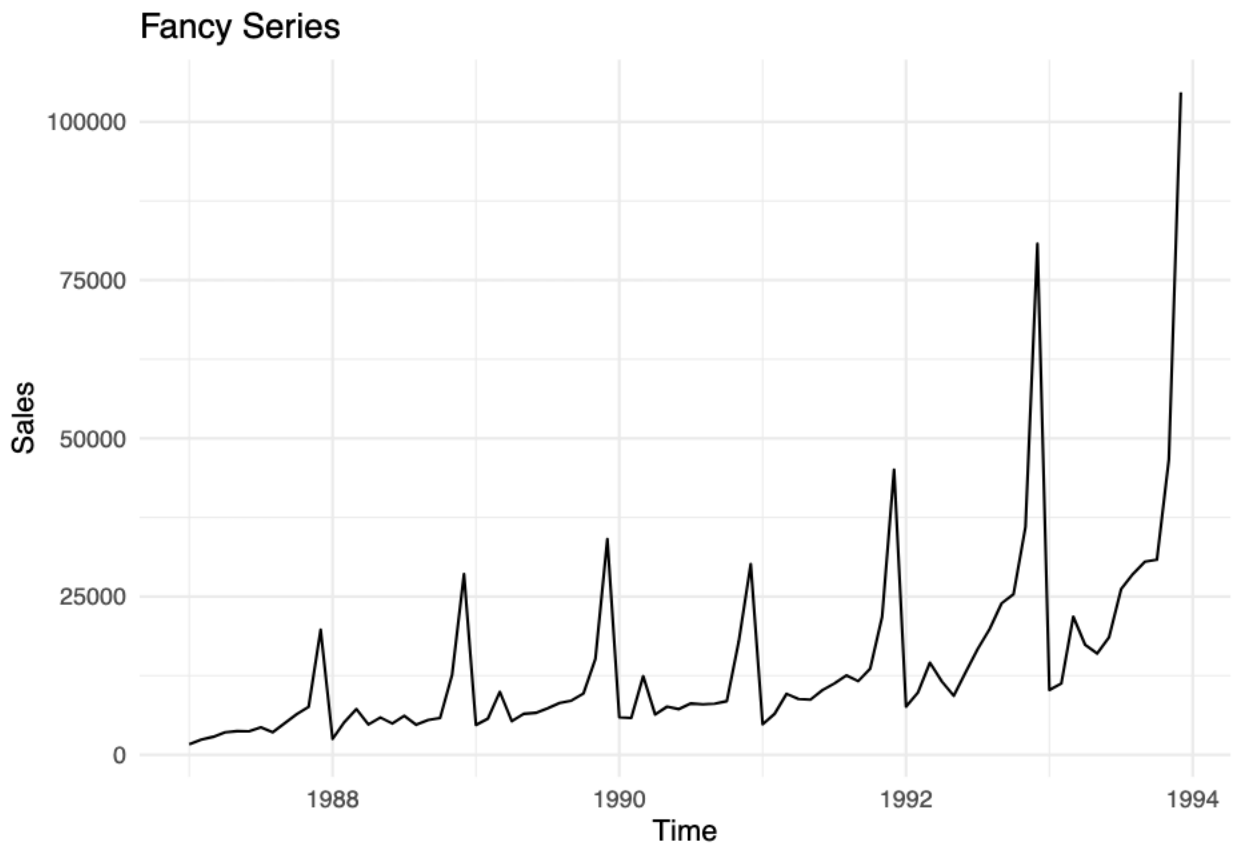

```
# Check residuals for autocorrelation and normality
checkresiduals(forecast)
```




```
##
##  Ljung-Box test
##
## data:  Residuals from STL +  Random walk with drift
## Q* = 68.38, df = 24, p-value = 3.837e-06
##
## Model df: 0.   Total lags used: 24
```

Problem 6.7 (i.e., Chapter 6, Problem 8) from Textbook (c)

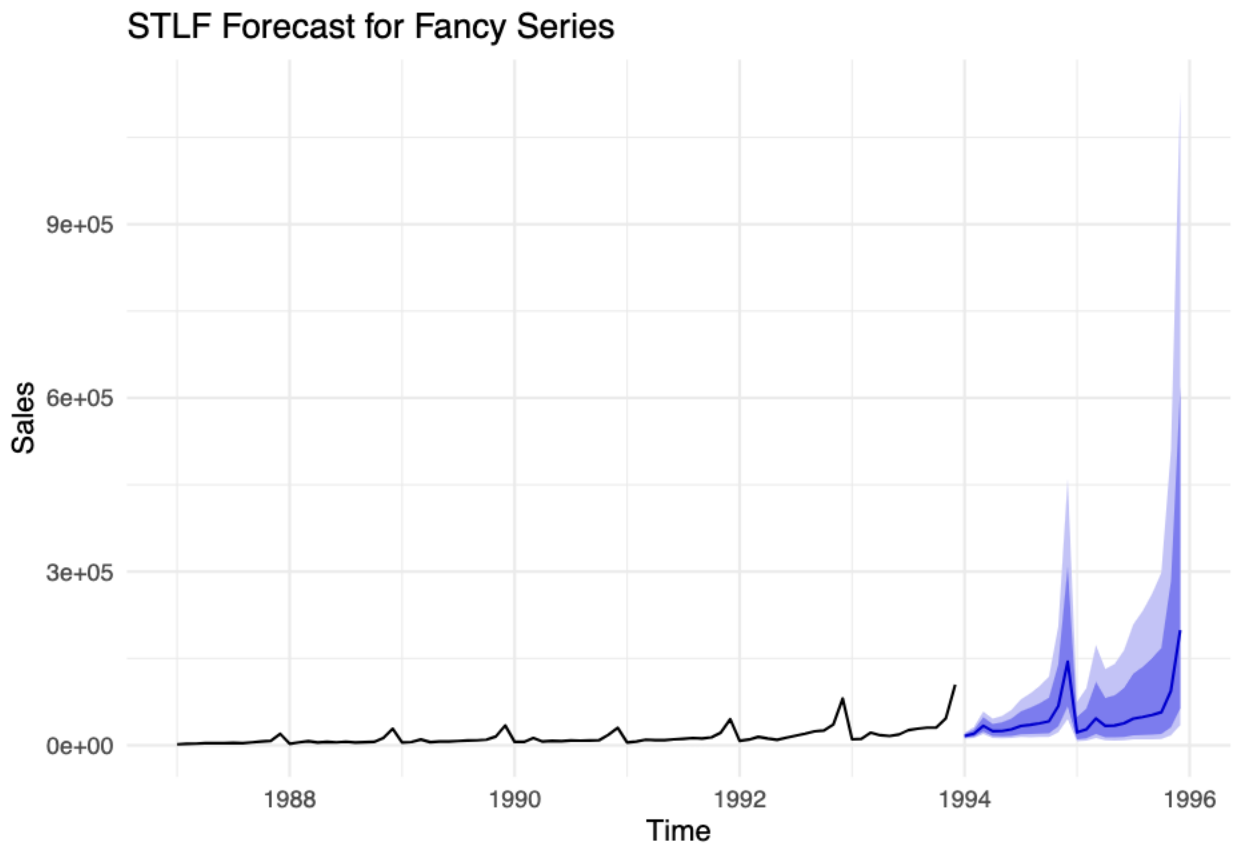
```r
#Load the fancy data
data("fancy")

#Plot the time series
autoplot(fancy) +
  labs(title = "Fancy Series", x = "Time", y = "Sales") +
  theme_minimal()
```



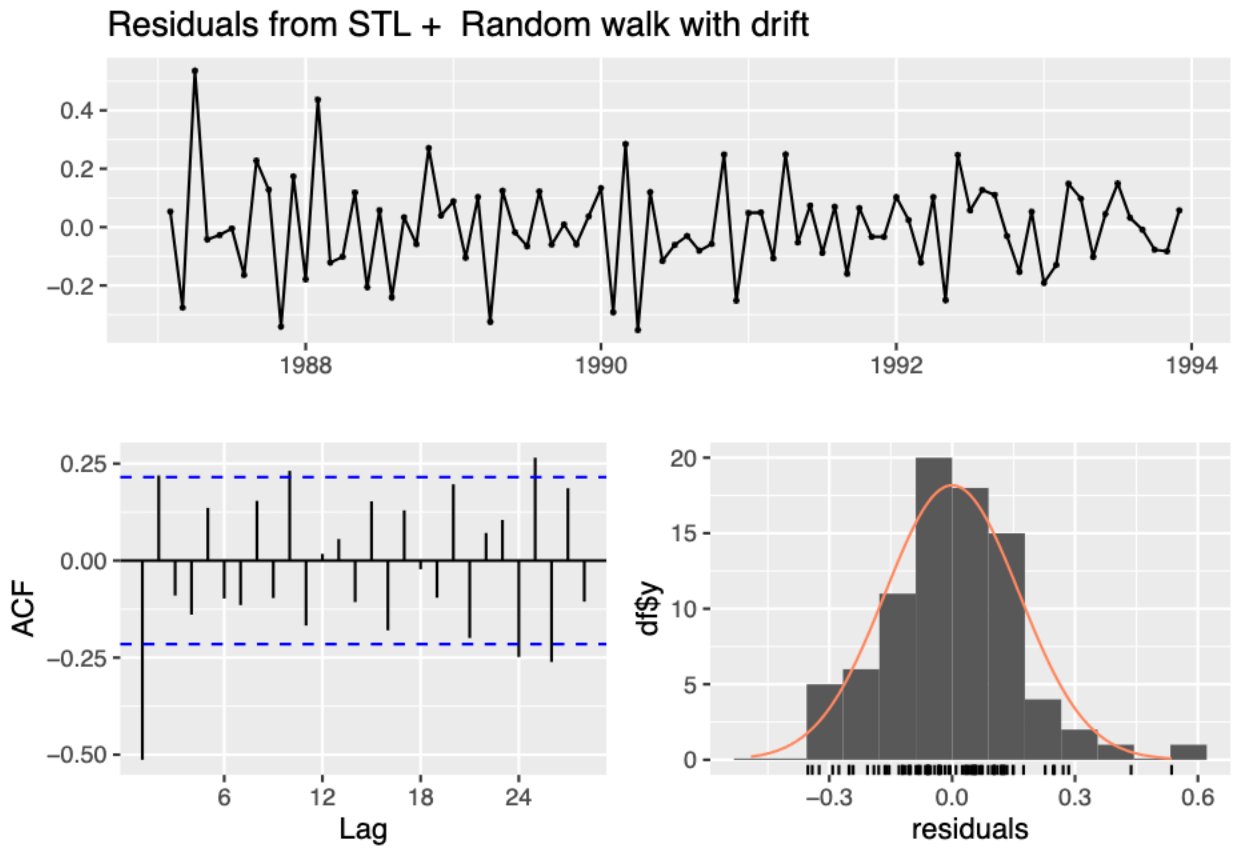

```
#Calculate lambda for potential Box-Cox tranformation
lambda <- BoxCox.lambda(fancy)
lambda
```

```
## [1] 0.002127317
```

```
# Use stlf() with Box-Cox transformation
fancy_forecast <- stlf(fancy, method = "rwdrift", lambda = lambda)

# Plot the forecast
autoplot(fancy_forecast) +
  labs(title = "STLF Forecast for Fancy Series", x = "Time", y = "Sales") +
  theme_minimal()
```




```
# Check residuals for autocorrelation and any patterns
checkresiduals(fancy_forecast)
```




```
##
##  Ljung-Box test
##
## data:  Residuals from STL +  Random walk with drift
## Q* = 53.205, df = 17, p-value = 1.326e-05
##
## Model df: 0.   Total lags used: 17
```