Objective:

Analyze the Consumer Price Index for All Urban Consumers: All Items in U.S. City Average (CPIAUCNS) from FRED, applying various time series techniques and models to forecast future values.

```r
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
```

```
## Loading required package: TTR
```

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

```r
library(dplyr)
```

```
##
## ######################### 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: 'dplyr'
```

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

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
library(ggplot2)
library(plotly)
```

```
##
## Attaching package: 'plotly'
```

```
## The following object is masked from 'package:ggplot2':
##
##     last_plot
```

```
## The following object is masked from 'package:stats':
##
##     filter
```

```
## The following object is masked from 'package:graphics':
##
##     layout
```

```
library(zoo)
library(xts)
library(stats)
library(gtrendsR)
library(quantmod)
library(lubridate)
```

```
##
## Attaching package: 'lubridate'
```

```
## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

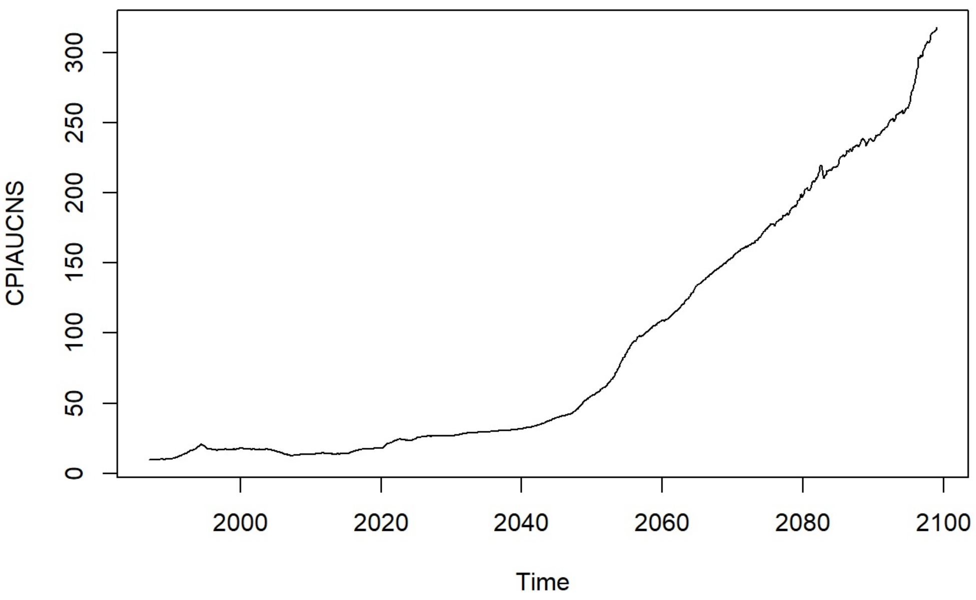
```
library(gapminder)
library(ROCR)
library(corrplot)
```

```
## corrplot 0.95 loaded
```

```
library(languageserver)
library(lubridate)
library(forecast)
library(TTR)
rm(list=ls())
source("https://bigblue.depaul.edu/jlee141/econdata/R/func_tslib.R")


# 1. Retrieve CPI date from FRED
getSymbols("CPIAUCNS", src = "FRED")
```

```
## [1] "CPIAUCNS"
```

```
# 2. Create a time series object
CPI <- coredata(CPIAUCNS)

# 3. Create a time series plot
ts_city_average <- ts(CPI, frequency = 12, start=c(1987,1)) # assuming the data is monthly
plot(ts_city_average)
```



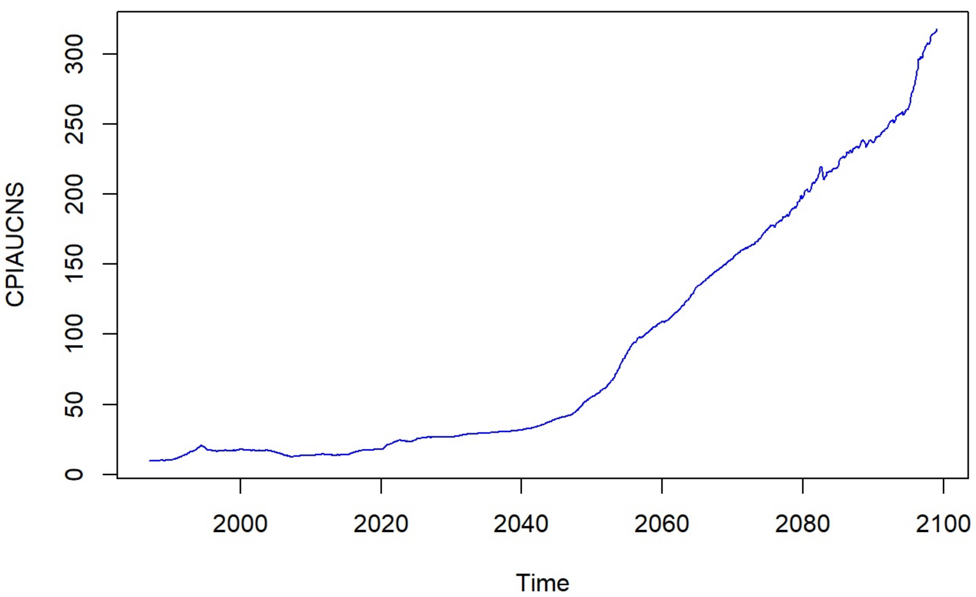
● Trends: Analyzing the time series plot, we observe fluctuations in the Consumer Price Index (CPI) over time. There appears to be an overall upward trend, indicating a general increase in consumer prices over the years.

● Seasonal Patterns: There might be seasonal patterns present in the CPI data, which can be identified by recurring patterns or fluctuations occurring at regular intervals within each year. However, these patterns are very minimal and not displayed in the graph.

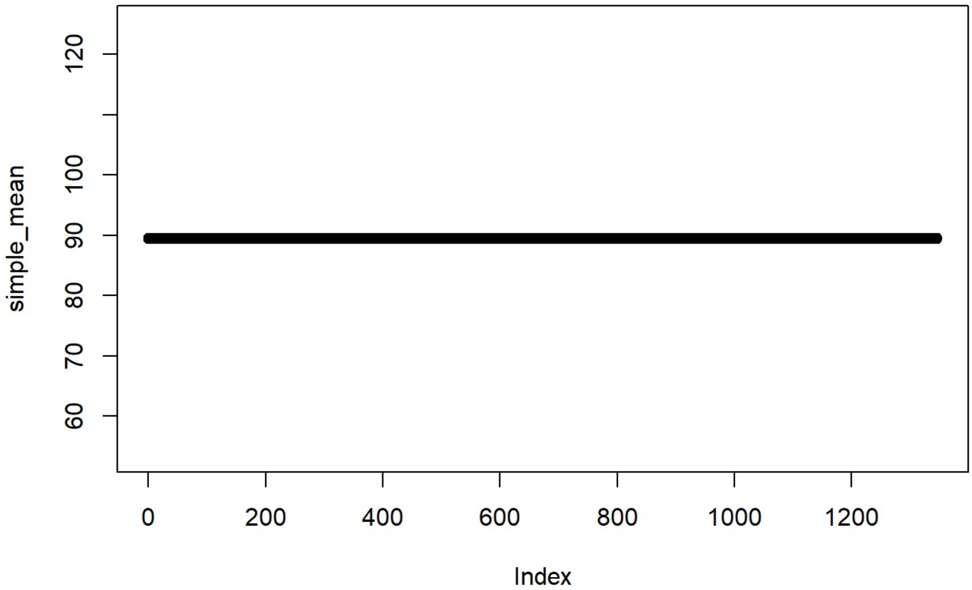
● Outliers: Potential outliers in the CPI data could manifest as sudden spikes or drops in the index that deviate significantly from the overall trend. In our data, there is only 1 events or economic shocks impacting consumer prices and should be investigated further to understand their underlying causes near the year 2084.

```
# 4. Decomposition and Basic Forecasting methods
# a. Seasonality and Trend Analysis
avg_decomp <- decompose(ts_city_average)
plot(avg_decomp$x, col="blue", main="Average")
```

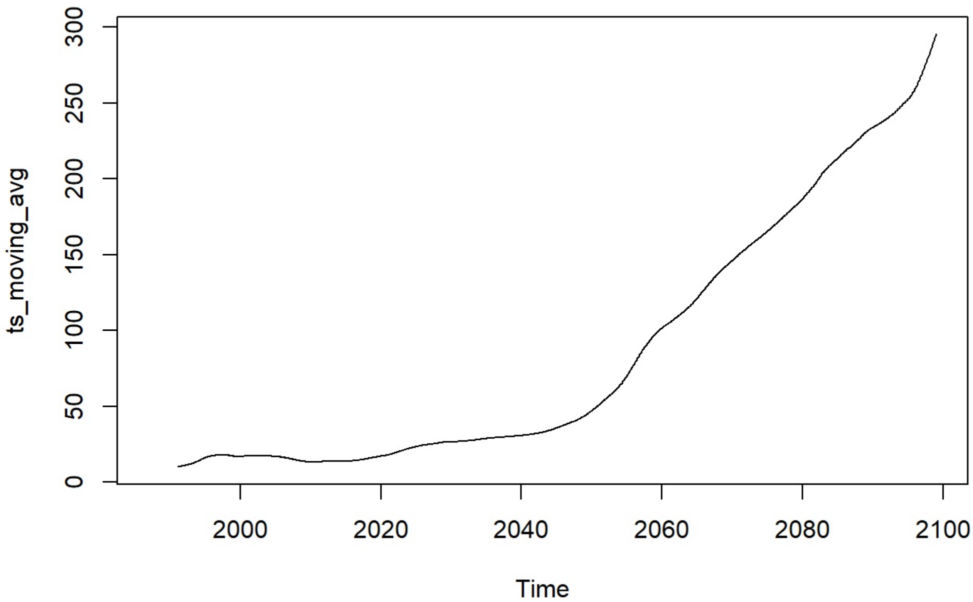
Average



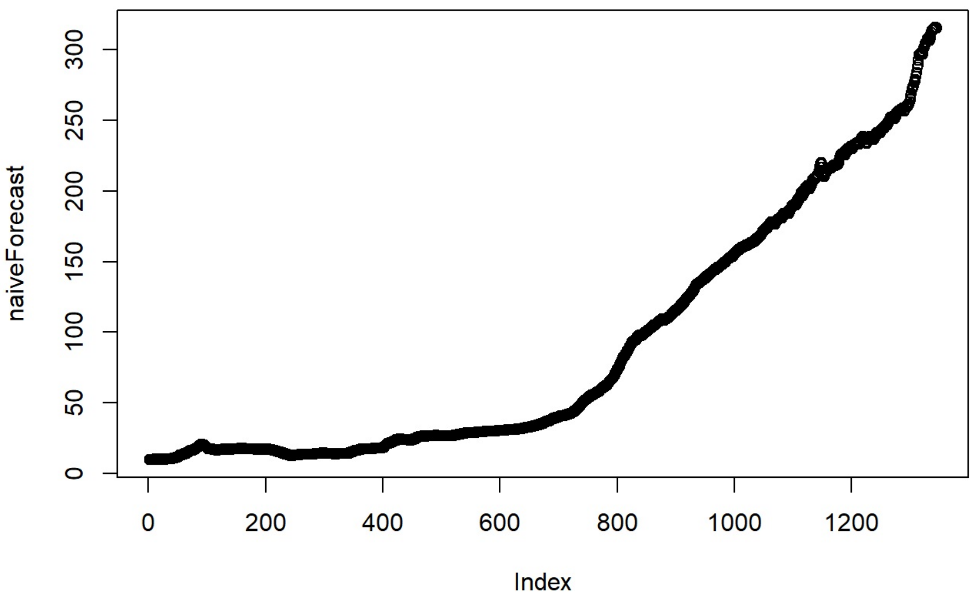

```
# b. Calculate and compare forecasting models
# Simple Mean Forecast
ts_mean <- mean(ts_city_average, na.rm = TRUE)
simple_mean <- rep(ts_mean,nrow(ts_city_average))
plot(simple_mean)
```



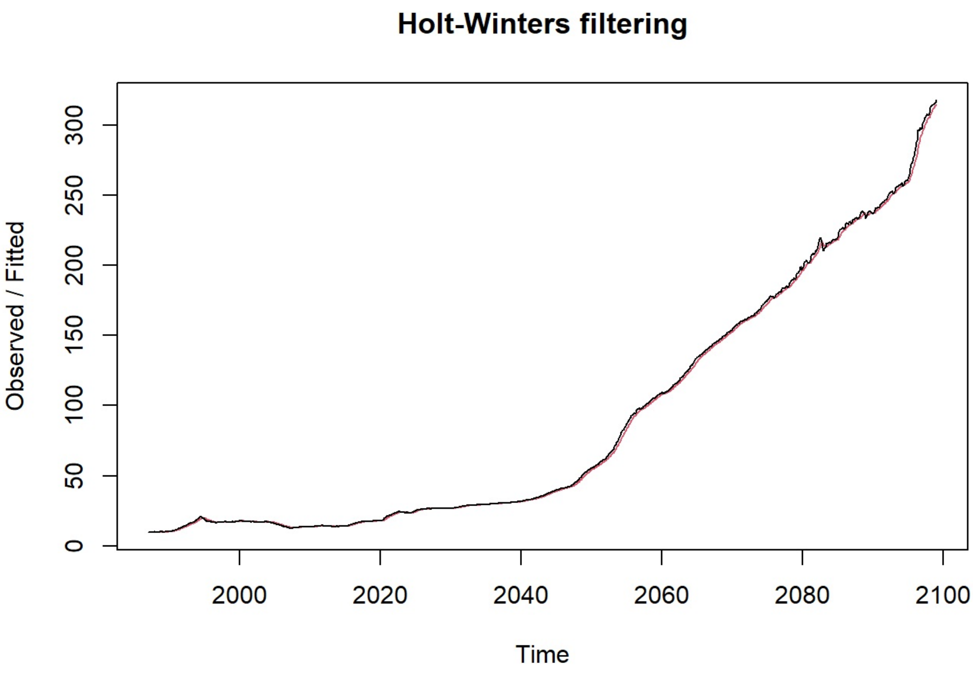

```
# Moving Average
n <- 50
ts_moving_avg <- SMA(ts_city_average, n = n)
plot(ts_moving_avg)
```



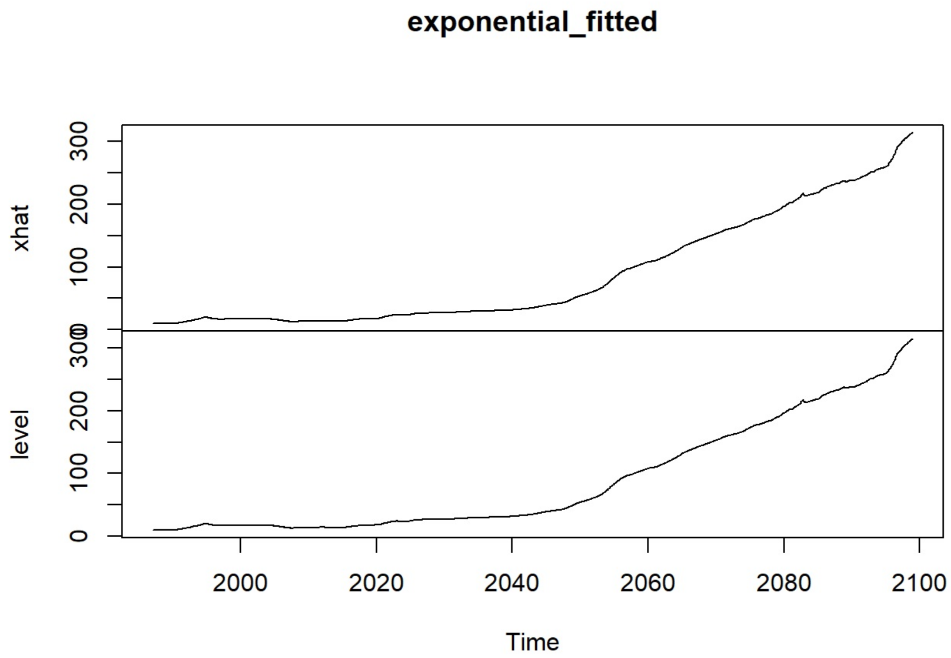

```
# Naive Forecast
naiveForecast <- lag(CPI, 1)
plot(naiveForecast)
```




```
# Exponential Smoothing
# Using HoltWinters() function
exponential_forecast <- HoltWinters(ts_city_average, alpha = 0.2, beta = FALSE, gamma = FALSE)
plot(exponential_forecast)
```




```
exponential_fitted <- fitted(exponential_forecast)
plot(exponential_fitted)
```



The evaluation of the forecasting methods suggests that the Simple Mean Forecast is the optimal choice for projecting future values of the Consumer Price Index (CPI). This method offers a direct prediction by calculating the average of past data, disregarding any underlying trends or cyclical variations.

It should be acknowledged that while the Simple Mean Forecast method offers ease of use, it might not effectively reflect intricate behaviors or shifts in consumer prices. To enhance the precision of forecasts, more sophisticated techniques like exponential smoothing or ARIMA models might be necessary, particularly for time series data that display trends, seasonal behaviors, or unpredictable variations.

Conducting Stationarity tests where:

Null Hypothesis (Ho): The series is stationary if the p-value is less than 0.05.

Alternative Hypothesis (H1): The series is not stationary if the p-value is greater than 0.05.

```
# 5. Stationarity test and Model identification
# a. Unit root test
unitroot_tests(CPIAUCNS)
```

```
## Loading required package: tseries
```

```
## Warning in adf.test(series, alternative = "stationary"): p-value greater than
## printed p-value
```

```
## Warning in kpss.test(series, null = "Level", lshort = TRUE): p-value smaller
## than printed p-value
```

```
## Warning in pp.test(series): p-value greater than printed p-value
```

```
##                              Test Lag_order  Statistic Stationary_P_Value
## Dickey-Fuller           ADF Test        11  0.9254178               0.99
## Dickey-Fuller Z(alpha)   PP Test         7  1.5290050               0.99
## KPSS Level             KPSS Test         7 14.8270164               0.01
```

```
# Ho = Series is stationary(Null Hypothesis). p-value < 0.05
# H1 = Series is not stationary(Alternative Hypothesis). p-value > 0.05
# For ADF and PP test, p-value should be less than 0.05 to reject the null hypothesis and for data to be stationa
ry.
# After analyzing the p-value for ADF and PP test, we can conclude that the data is not stationary since p-values
are greater than 0.05.

#  b. Achieving Stationarity
# Use differencing to make the data stationary
diff_CPI <- diff(CPI, lag=1)
unitroot_tests(diff(CPI, lag=1))
```

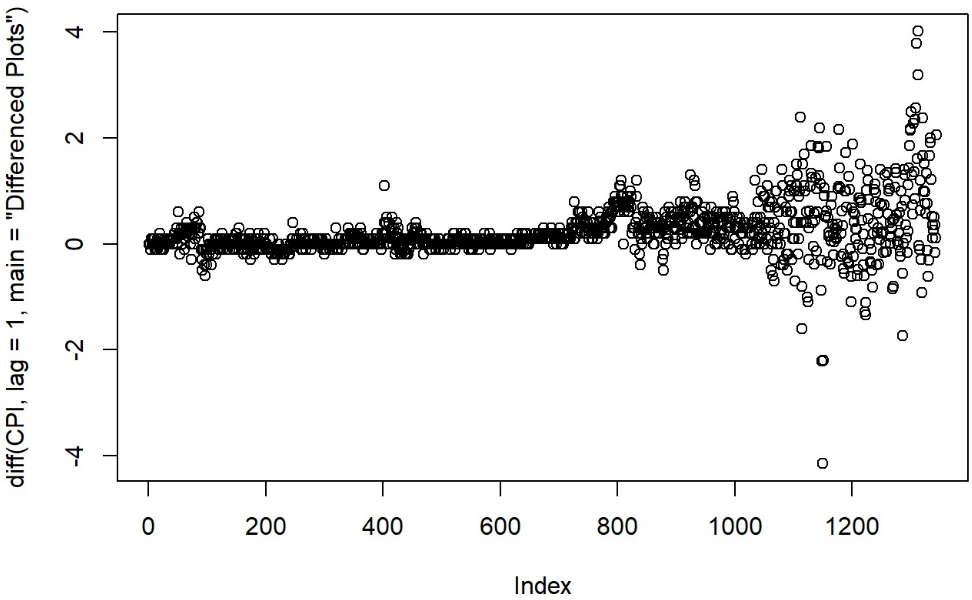
```
## Warning in adf.test(series, alternative = "stationary"): p-value smaller than
## printed p-value
```

```
## Warning in kpss.test(series, null = "Level", lshort = TRUE): p-value smaller
## than printed p-value
```

```
## Warning in pp.test(series): p-value smaller than printed p-value
```

```
##                               Test Lag_order   Statistic Stationary_P_Value
## Dickey-Fuller             ADF Test        11   -5.697769               0.01
## Dickey-Fuller Z(alpha)     PP Test         7 -572.228888               0.01
## KPSS Level               KPSS Test         7    6.429639               0.01
```

```
plot(diff(CPI, lag=1, main="Differenced Plots"))
```

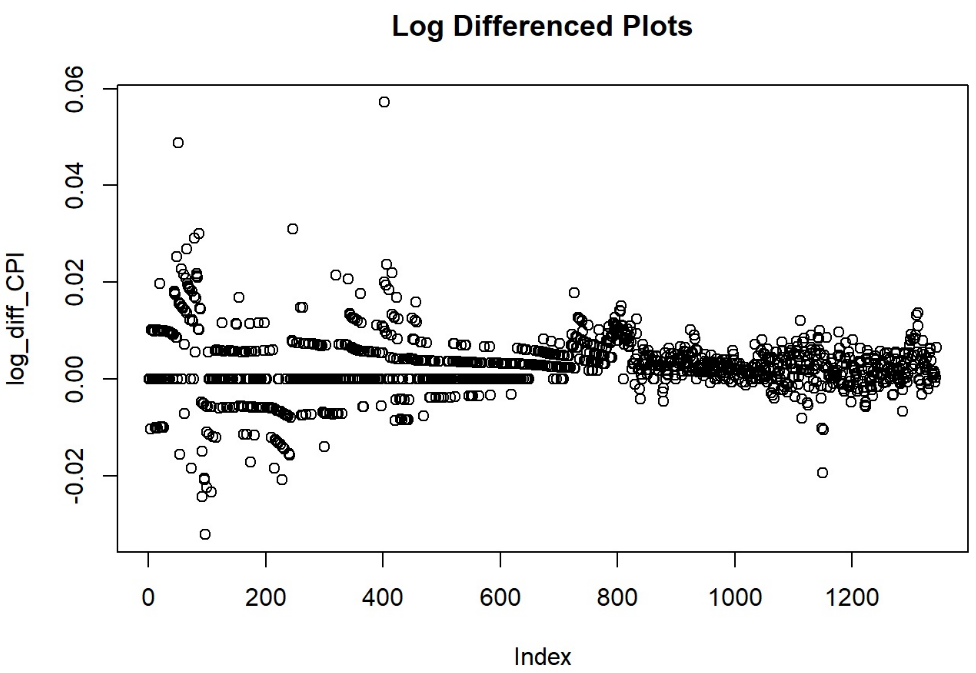



```
# After differcing the data, unit root test shows that the data is stationary.
# p-value for ADF and PP test are less than 0.05
# Hence, data is stationary.

# c. Using log differencing
log_diff_CPI <- diff(log(CPI), lag=1)
unitroot_tests(log_diff_CPI)
```

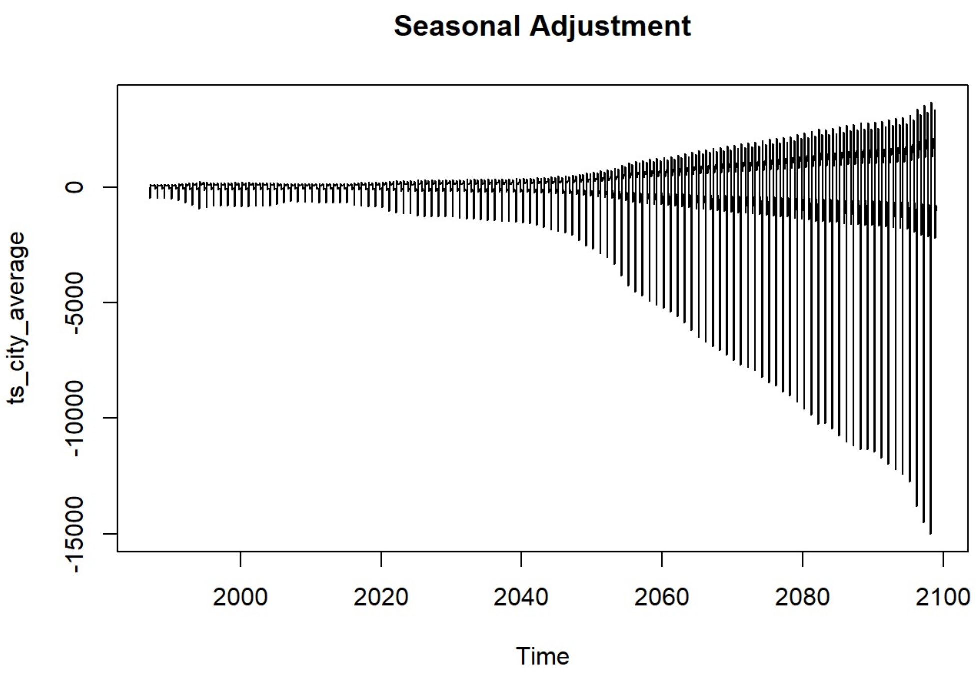
```
## Warning in adf.test(series, alternative = "stationary"): p-value smaller than
## printed p-value
## Warning in adf.test(series, alternative = "stationary"): p-value smaller than
## printed p-value
```

```
##                               Test Lag_order   Statistic Stationary_P_Value
## Dickey-Fuller             ADF Test        11   -5.541913         0.01000000
## Dickey-Fuller Z(alpha)     PP Test         7 -912.259382         0.01000000
## KPSS Level               KPSS Test         7    0.360515         0.09417455
```

```
plot(log_diff_CPI, main="Log Differenced Plots")
```



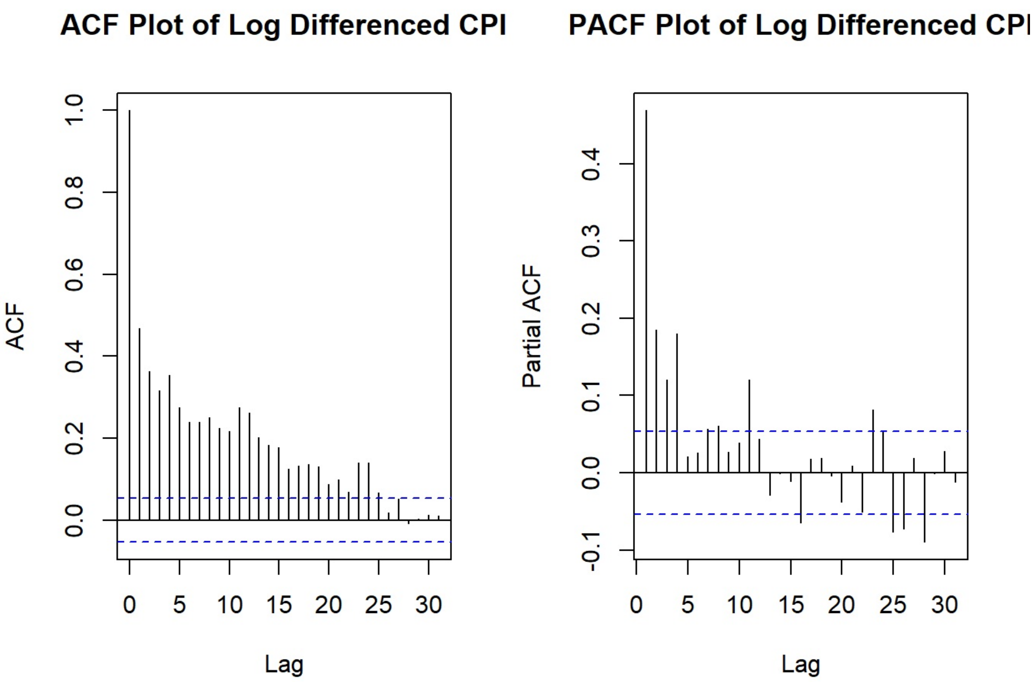

```
# After differcing the data, unit root test shows that the data is stationary.
# p-value for ADF and PP test are less than 0.05
# Hence, data is stationary.

# c. Seasonal Adjustment
adjusted_ts <- ts_city_average / avg_decomp$seasonal
plot(adjusted_ts, main="Seasonal Adjustment")
```



Seasonal Adjustment trend indicates an increase in the percentage of seasonal adjustment as time progresses. Hence, the graph visually shows how the seasonal adjustment changes over the years, with a clear upward trend.

3. ACF and PACF Analysis

```
# 6. ACG and PACF plots
par(mfrow =c(1,2))
acf(log_diff_CPI, main="ACF Plot of Log Differenced CPI")
pacf(log_diff_CPI, main = "PACF Plot of Log Differenced CPI")
```




```
# According the ACF and PACF plots, data is Autoregressive (AR) model
# Here, Model has a high autocorrelation as seen from ACF plot.
# This means that the current value of a variable depends on its previous values.
# PACF plot shows that the model has a low correlation with the past values.
# Order of AR(4)
```

• Stationarity Achieved: The unit root tests have verified that the differenced and log differenced CPI data are now stationary, signifying the effective removal of trends, seasonal effects, and other non-stationary elements.

• Insightful Analysis: The tests confirming stationarity and the process of model selection offer critical guidance in choosing a fitting time series model, paving the way for precise forecasts of future CPI figures.

• Model Type: Based on the ACF and PACF plots, the data follows an Autoregressive (AR) model.

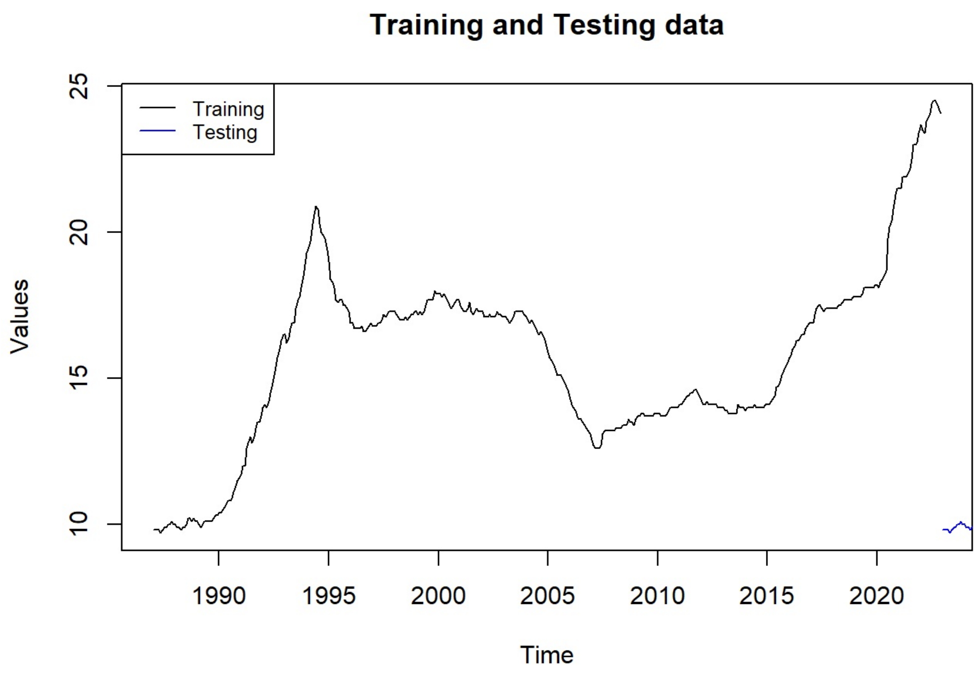
• High Autocorrelation: The ACF plot reveals a significant autocorrelation, indicating that current values are closely related to their preceding values.

• Low Partial Correlation: The PACF plot suggests minimal correlation with past values beyond the immediate one.

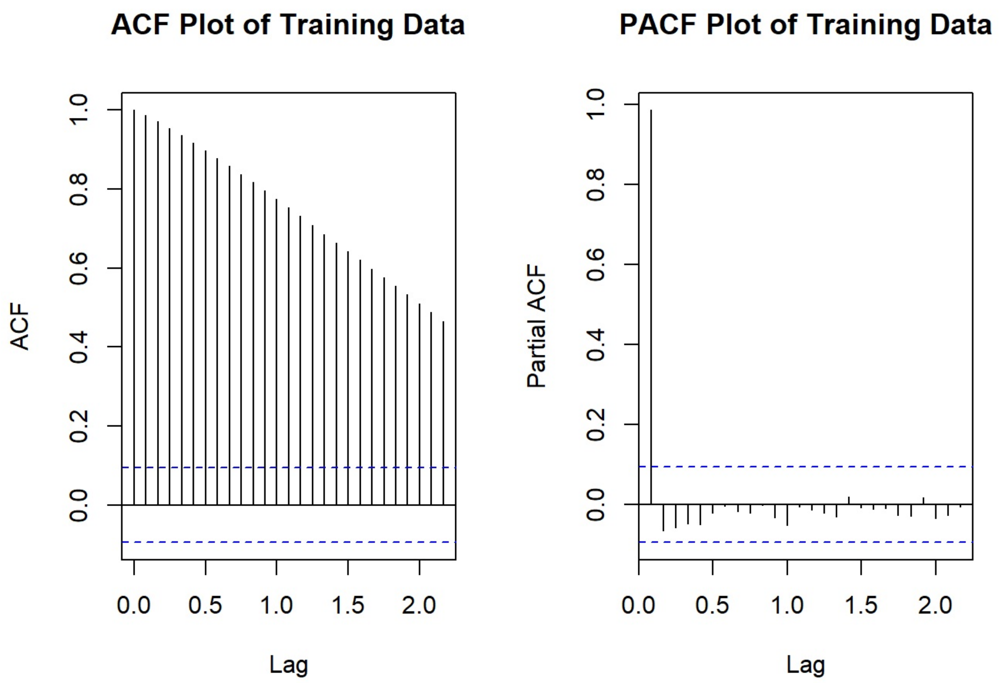
• Model Order: The appropriate order for the AR model is 4, denoted as AR(4).

1. Model Development

```
# Part 4 : Model Building and Evaluation
# 1. Model Development
# Splitting data into training and testing
# Converting data to time series
CPI_training_base <- ts(CPIAUCNS, frequency = 12, start=c(1987,1), end=c(2022,12))
CPI_testing_base <- ts(CPIAUCNS, frequency = 12, start=c(2023,1))

# Plotting training and testing data for visual representation
par(mfrow =c(1,1))
plot(CPI_training_base, main="Training and Testing data", xlab = "Time", ylab="Values")
lines(CPI_testing_base, col="blue")
legend("topleft", legend=c("Training", "Testing"), col=c("black", "blue"), lty=1, cex=0.8)
```




```
# Implementing ACF and PACF plots
par(mfrow =c(1,2))
acf(CPI_training_base, main = "ACF Plot of Training Data")
pacf(CPI_training_base, main = "PACF Plot of Training Data")
```



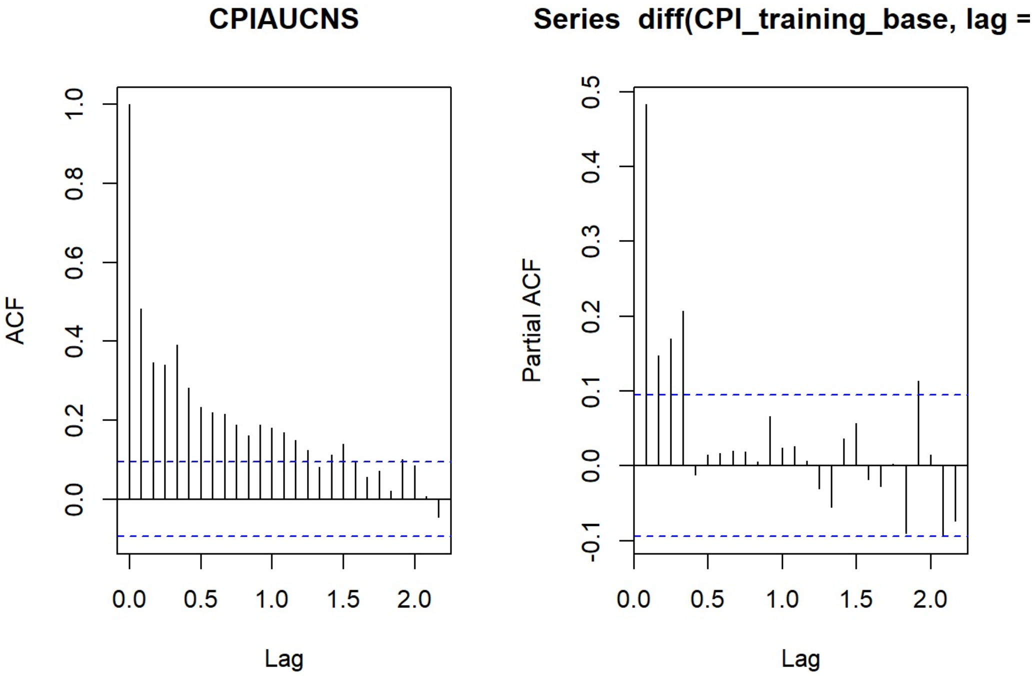

```
unitroot_tests(CPI_training_base)
```

```
## Warning in kpss.test(series, null = "Level", lshort = TRUE): p-value smaller
## than printed p-value
```

```
## Warning in pp.test(series): p-value greater than printed p-value
```

```
##                             Test Lag_order  Statistic Stationary_P_Value
## Dickey-Fuller           ADF Test         7 -1.6088711          0.7429252
## Dickey-Fuller Z(alpha)   PP Test         5 -0.7460705          0.9900000
## KPSS Level             KPSS Test         5  1.6422904          0.0100000
```

```
# According the ACF and PACF plots, data is Autoregressive (AR) model
# Data is not stationary.

# Making the data stationary
acf(diff(CPI_training_base, lag=1))
pacf(diff(CPI_training_base, lag=1))
```




```
unitroot_tests(diff(CPI_training_base, lag=1))
```

```
## Warning in adf.test(series, alternative = "stationary"): p-value smaller than
## printed p-value
```

```
## Warning in pp.test(series): p-value smaller than printed p-value
```

```
##                                Test Lag_order    Statistic Stationary_P_Value
## Dickey-Fuller           ADF Test         7   -4.5064945          0.0100000
## Dickey-Fuller Z(alpha)   PP Test         5 -255.1870653          0.0100000
## KPSS Level             KPSS Test         5    0.4796536          0.0462492
```

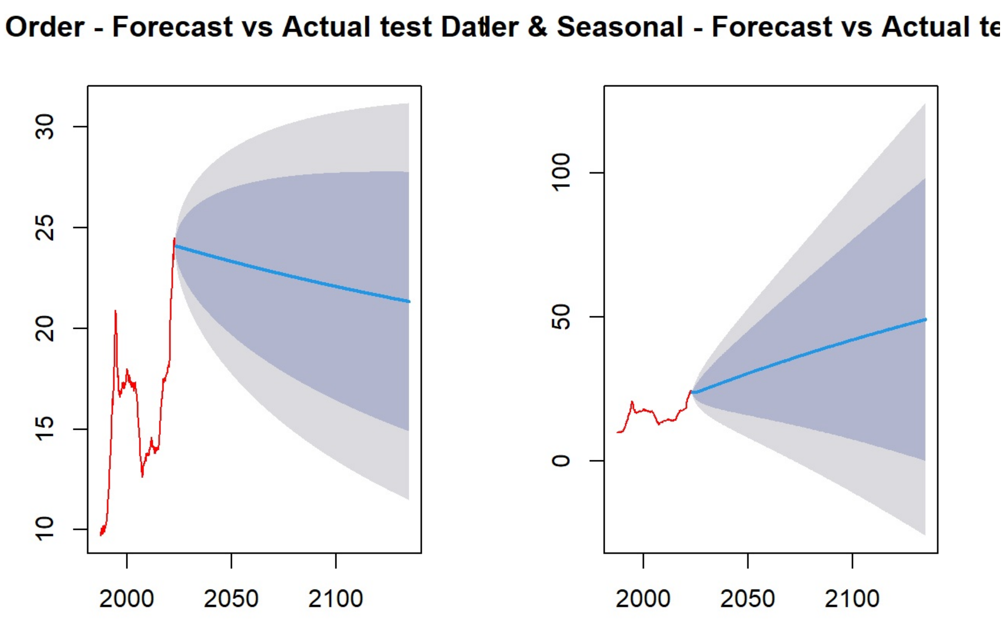
```
# According the ACF and PACF plots, data is Autoregressive (AR) model
# Data is stationary with order of 4. AR(4)

# Developing different models
# Fit the ARIMA model using MLE
model1 <- arima(CPI_training_base, order=c(1,0,0), method = "ML")
summary(model1)
```

```
##
## Call:
## arima(x = CPI_training_base, order = c(1, 0, 0), method = "ML")
##
## Coefficients:
##          ar1  intercept
##       0.9997    15.8515
## s.e.  0.0004     6.6696
##
## sigma^2 estimated as 0.0275:  log likelihood = 159.55,  aic = -313.1
##
## Training set error measures:
##                     ME      RMSE       MAE       MPE      MAPE     MASE
## Training set 0.0327242 0.1658214 0.1100714 0.1982416 0.6884411 1.002976
##                   ACF1
## Training set 0.4823841
```

```
forecast1 <- forecast(model1, h=length(CPI_testing_base))
plot(forecast1, main = "Order - Forecast vs Actual test Data", col="red")

model2 <- arima(CPI_training_base, order=c(4,1,1), seasonal = list(order=c(1,0,1), period=12), method = "ML")
summary(model2)
```

```
##
## Call:
## arima(x = CPI_training_base, order = c(4, 1, 1), seasonal = list(order = c(1,
##     0, 1), period = 12), method = "ML")
##
## Coefficients:
##          ar1     ar2     ar3     ar4     ma1    sar1     sma1
##       0.3230  0.0687  0.0989  0.2398  0.0142  0.9966  -0.9772
## s.e.  0.2024  0.0900  0.0517  0.0593  0.2089  0.0137   0.0482
##
## sigma^2 estimated as 0.01751:  log likelihood = 255.69,  aic = -495.38
##
## Training set error measures:
##                       ME      RMSE        MAE        MPE      MAPE      MASE
## Training set 0.004292515 0.132175 0.09295621 0.03534302 0.5884888 0.8470217
##                       ACF1
## Training set -0.0002299574
```

```
forecast2 <- forecast(model2, h=length(CPI_testing_base))
plot(forecast2, main = "Order & Seasonal - Forecast vs Actual test Data", col="red")
```




```
model3 <- auto.arima(CPI_training_base, seasonal = TRUE, D=0, max.P = 1,
    max.Q = 1, stepwise = FALSE, approximation = FALSE)
summary(model3)
```

```
## Series: CPI_training_base
## ARIMA(0,2,5)
##
## Coefficients:
##           ma1      ma2      ma3     ma4      ma5
##       -0.6347  -0.1600  -0.0029  0.1262  -0.1613
## s.e.   0.0486   0.0564   0.0596  0.0556   0.0496
##
## sigma^2 = 0.01907:  log likelihood = 243.16
## AIC=-474.32   AICc=-474.13   BIC=-449.94
##
## Training set error measures:
##                        ME      RMSE        MAE        MPE      MAPE      MASE
## Training set 0.0006158696 0.1369724 0.09702293 0.01966607 0.6114062 0.1067024
##                     ACF1
## Training set 0.008404599
```

```r
forecast3 <- forecast(model3, h=length(CPI_testing_base))
plot(forecast3, main = "Auto ARIMA - Forecast vs Actual test Data", col="red")

accuracy(model1)
```

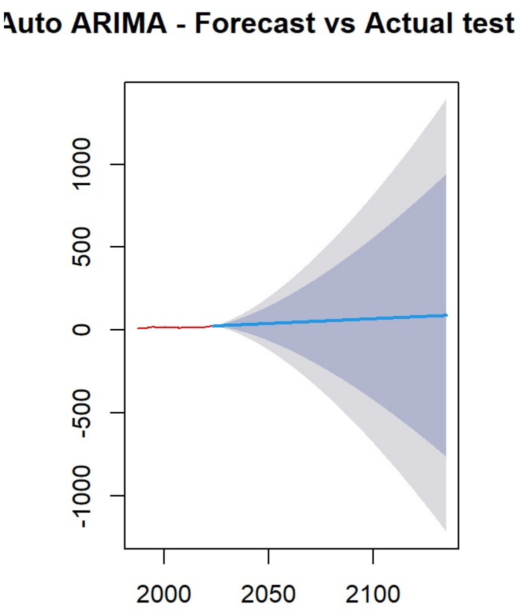
```
##                      ME      RMSE       MAE       MPE      MAPE     MASE
## Training set 0.0327242 0.1658214 0.1100714 0.1982416 0.6884411 1.002976
##                    ACF1
## Training set 0.4823841
```

```r
accuracy(model2)
```

```
##                       ME     RMSE        MAE        MPE      MAPE      MASE
## Training set 0.004292515 0.132175 0.09295621 0.03534302 0.5884888 0.8470217
##                       ACF1
## Training set -0.0002299574
```

```r
accuracy(model3)
```

```
##                        ME      RMSE        MAE        MPE      MAPE      MASE
## Training set 0.0006158696 0.1369724 0.09702293 0.01966607 0.6114062 0.1067024
##                      ACF1
## Training set 0.008404599
```



• Model Type: The unit root tests confirm that the differenced and log-differenced CPI data are now stationary. Based on the Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF) plots, the data follows an Autoregressive (AR) model.

• Autocorrelation: The ACF plot reveals significant autocorrelation, indicating that current values are closely related to their preceding values.

• Partial Correlation: The PACF plot suggests minimal correlation with past values beyond the immediate lag.

• Model Order: The appropriate order for the AR model is 4, denoted as AR(4).

Finding the best fitting model on test data

```r
# Defining the range for parameters
p_range <- 0:3
d_range <- 1:1
q_range <- 0:3
P_range <- 0:1
Q_range <- 0:1

best_rmse <- Inf
best_model <- NULL

for(p in p_range){
  for(d in d_range){
    for(q in q_range){
      for(P in P_range){
        for(Q in Q_range){
        # Fit model
        print(paste(p,d,q,P,Q))
        model <- Arima(CPI_training_base, order=c(p,d,q), seasonal = list(order = c(P, 0, Q), period = 12))
        # Forecast
        forecasts <- forecast(model, h=length(CPI_testing_base))
        # Calculate RMSE
        rmse <- sqrt(mean((forecasts$mean - CPI_testing_base)^2))
        print(paste(p,d,q,P,Q,rmse))

        # Checking if model is better
        if(rmse < best_rmse) {
            best_rmse <- rmse
            best_model <- model
            cat(sprintf("New best model found: ARIMA(%s, %s, %s)(%s, 0, %s) with RMSE: %f\n", p,d,q,P,Q,rmse))
        }
        }
        }
    }
  }
}
```

```
## [1] "0 1 0 0 0"
## [1] "0 1 0 0 0 109.395704735362"
## New best model found: ARIMA(0, 1, 0)(0, 0, 0) with RMSE: 109.395705
## [1] "0 1 0 0 1"
## [1] "0 1 0 0 1 109.345115694381"
## New best model found: ARIMA(0, 1, 0)(0, 0, 1) with RMSE: 109.345116
## [1] "0 1 0 1 0"
## [1] "0 1 0 1 0 109.28039284536"
## New best model found: ARIMA(0, 1, 0)(1, 0, 0) with RMSE: 109.280393
## [1] "0 1 0 1 1"
## [1] "0 1 0 1 1 108.638212062976"
## New best model found: ARIMA(0, 1, 0)(1, 0, 1) with RMSE: 108.638212
## [1] "0 1 1 0 0"
## [1] "0 1 1 0 0 109.404341418177"
## [1] "0 1 1 0 1"
## [1] "0 1 1 0 1 109.375279672228"
## [1] "0 1 1 1 0"
## [1] "0 1 1 1 0 109.348001197644"
## [1] "0 1 1 1 1"
## [1] "0 1 1 1 1 107.991384387352"
## New best model found: ARIMA(0, 1, 1)(1, 0, 1) with RMSE: 107.991384
## [1] "0 1 2 0 0"
## [1] "0 1 2 0 0 109.419684069726"
## [1] "0 1 2 0 1"
## [1] "0 1 2 0 1 109.403285188033"
## [1] "0 1 2 1 0"
## [1] "0 1 2 1 0 109.386224832825"
## [1] "0 1 2 1 1"
## [1] "0 1 2 1 1 108.170034518635"
## [1] "0 1 3 0 0"
## [1] "0 1 3 0 0 109.433207697826"
## [1] "0 1 3 0 1"
## [1] "0 1 3 0 1 109.417467909801"
## [1] "0 1 3 1 0"
## [1] "0 1 3 1 0 109.402687141462"
## [1] "0 1 3 1 1"
## [1] "0 1 3 1 1 108.159407719046"
## [1] "1 1 0 0 0"
## [1] "1 1 0 0 0 109.456216016764"
## [1] "1 1 0 0 1"
## [1] "1 1 0 0 1 109.445070860385"
```

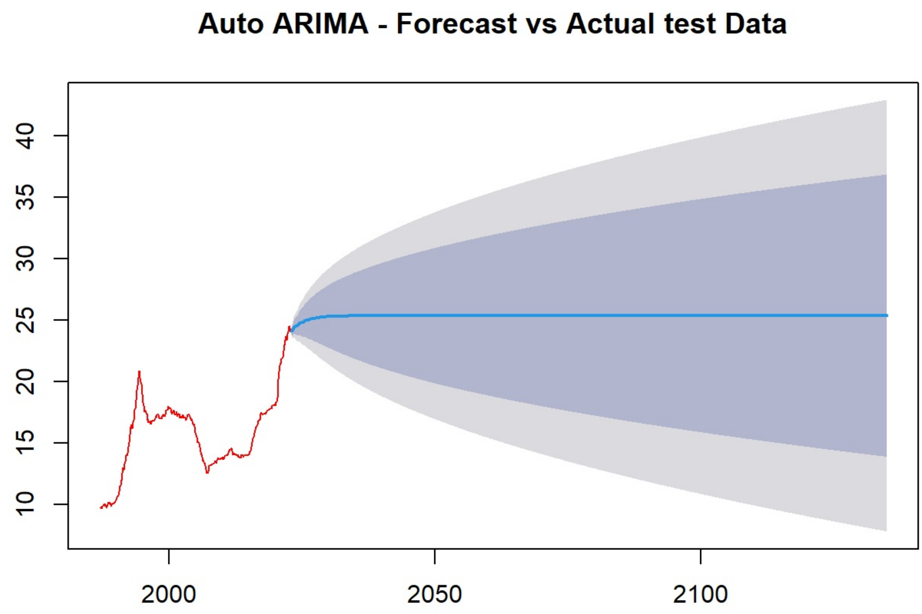
```
## [1] "1 1 0 1 0"
## [1] "1 1 0 1 0 109.436178100102"
## [1] "1 1 0 1 1"
## [1] "1 1 0 1 1 96.5168542779068"
## New best model found: ARIMA(1, 1, 0)(1, 0, 1) with RMSE: 96.516854
## [1] "1 1 1 0 0"
## [1] "1 1 1 0 0 109.738338118886"
## [1] "1 1 1 0 1"
## [1] "1 1 1 0 1 109.753290159542"
## [1] "1 1 1 1 0"
## [1] "1 1 1 1 0 109.756551346741"
## [1] "1 1 1 1 1"
## [1] "1 1 1 1 1 97.6720700861881"
## [1] "1 1 2 0 0"
## [1] "1 1 2 0 0 109.603564997971"
## [1] "1 1 2 0 1"
## [1] "1 1 2 0 1 109.628677411076"
## [1] "1 1 2 1 0"
## [1] "1 1 2 1 0 109.63326051388"
## [1] "1 1 2 1 1"
## [1] "1 1 2 1 1 97.0664435426209"
## [1] "1 1 3 0 0"
## [1] "1 1 3 0 0 109.688553621108"
## [1] "1 1 3 0 1"
## [1] "1 1 3 0 1 109.719831609466"
## [1] "1 1 3 1 0"
## [1] "1 1 3 1 0 109.7254661796"
## [1] "1 1 3 1 1"
## [1] "1 1 3 1 1 96.6658126297102"
## [1] "2 1 0 0 0"
## [1] "2 1 0 0 0 109.525212342896"
## [1] "2 1 0 0 1"
## [1] "2 1 0 0 1 109.520429235646"
## [1] "2 1 0 1 0"
## [1] "2 1 0 1 0 109.51537294923"
## [1] "2 1 0 1 1"
## [1] "2 1 0 1 1 97.0446316021269"
## [1] "2 1 1 0 0"
## [1] "2 1 1 0 0 109.600086990189"
## [1] "2 1 1 0 1"
## [1] "2 1 1 0 1 109.629020455688"
## [1] "2 1 1 1 0"
## [1] "2 1 1 1 0 109.634516317261"
## [1] "2 1 1 1 1"
## [1] "2 1 1 1 1 97.3320107497982"
## [1] "2 1 2 0 0"
## [1] "2 1 2 0 0 109.628038043885"
## [1] "2 1 2 0 1"
## [1] "2 1 2 0 1 109.653056730117"
## [1] "2 1 2 1 0"
## [1] "2 1 2 1 0 109.658050263576"
## [1] "2 1 2 1 1"
## [1] "2 1 2 1 1 96.8513999766157"
## [1] "2 1 3 0 0"
## [1] "2 1 3 0 0 109.546097821006"
## [1] "2 1 3 0 1"
## [1] "2 1 3 0 1 109.57060807401"
## [1] "2 1 3 1 0"
## [1] "2 1 3 1 0 109.575655894092"
## [1] "2 1 3 1 1"
## [1] "2 1 3 1 1 97.0498919338383"
## [1] "3 1 0 0 0"
## [1] "3 1 0 0 0 109.634005729405"
## [1] "3 1 0 0 1"
## [1] "3 1 0 0 1 109.635242058089"
## [1] "3 1 0 1 0"
## [1] "3 1 0 1 0 109.633642181441"
## [1] "3 1 0 1 1"
## [1] "3 1 0 1 1 96.7767966654113"
## [1] "3 1 1 0 0"
## [1] "3 1 1 0 0 109.730192761549"
## [1] "3 1 1 0 1"
## [1] "3 1 1 0 1 109.751948538678"
## [1] "3 1 1 1 0"
## [1] "3 1 1 1 0 109.755618163432"
## [1] "3 1 1 1 1"
## [1] "3 1 1 1 1 96.5151190441447"
## New best model found: ARIMA(3, 1, 1)(1, 0, 1) with RMSE: 96.515119
## [1] "3 1 2 0 0"
```

```
## [1] "3 1 2 0 0 109.795016589207"
## [1] "3 1 2 0 1"
## [1] "3 1 2 0 1 109.813907619976"
## [1] "3 1 2 1 0"
## [1] "3 1 2 1 0 109.816765920204"
## [1] "3 1 2 1 1"
## [1] "3 1 2 1 1 96.7128013680102"
## [1] "3 1 3 0 0"
## [1] "3 1 3 0 0 109.634435902739"
## [1] "3 1 3 0 1"
## [1] "3 1 3 0 1 109.777292951451"
## [1] "3 1 3 1 0"
## [1] "3 1 3 1 0 109.783638441162"
## [1] "3 1 3 1 1"
## [1] "3 1 3 1 1 96.7135367997042"
```

```
# Best Model
summary(best_model)
```

```
## Series: CPI_training_base
## ARIMA(3,1,1)(1,0,1)[12]
##
## Coefficients:
##          ar1      ar2     ar3      ma1     sar1     sma1
##       0.9056  -0.1484  0.1361  -0.5692  0.9978  -0.9832
## s.e.  0.1142   0.0730  0.0589   0.1084  0.0082   0.0322
##
## sigma^2 = 0.01813:  log likelihood = 251.19
## AIC=-488.38   AICc=-488.11   BIC=-459.91
##
## Training set error measures:
##                       ME      RMSE        MAE        MPE      MAPE      MASE
## Training set 0.004219338 0.1335347 0.09420299 0.03551948 0.5957033 0.1036011
##                      ACF1
## Training set -0.008504367
```

```
Best_model <- arima(CPI_training_base, order=c(0,1,0), seasonal = list(order = c(1, 0, 1), period =12))

forecast4 <- forecast(Best_model, h=length(CPI_testing_base))
plot(forecast4, main = "Auto ARIMA - Forecast vs Actual test Data", col="red")
```




```
# Summary of performance models
table <- rbind(accuracy(forecast1$mean, CPI_testing_base),
               accuracy(forecast2$mean, CPI_testing_base),
               accuracy(forecast3$mean, CPI_testing_base),
               accuracy(forecast4$mean, CPI_testing_base))
row.names(table) <- c("Arima Model with Order", "Arima Model with Order & Seasonal", "Auto ARIMA", "Grid Search")
print(table)
```

```
##                                         ME      RMSE      MAE         MPE
## Arima Model with Order            66.82803 110.85392 71.86625  25.6265761
## Arima Model with Order & Seasonal 52.44701  96.60490 62.56018   0.6709523
## Auto ARIMA                        33.32304  78.54982 54.77888 -32.5451702
## Grid Search                       64.10496 108.63821 70.27462  19.1153496
##                                       MAPE      ACF1 Theil's U
## Arima Model with Order            61.83722 0.9971646  100.7887
## Arima Model with Order & Seasonal 62.39561 0.9970884  102.0745
## Auto ARIMA                        77.54233 0.9969804  128.1240
## Grid Search                       62.45307 0.9971665  102.6052
```

Summary

● Based on the model evaluations and forecasting results, the ARIMA(0,1,0)(1,0,1)[12] model outperforms other models in terms of forecast accuracy, as it has the lowest RMSE on the testing data.

● The forecasting plots visually demonstrate the performance of each model in capturing the CPI trends and variations.

● The grid search approach helps in systematically selecting the best ARIMA model by considering various parameter combinations.

● Overall, the selected ARIMA model can be used to forecast future CPI values with reasonable accuracy, providing valuable insights for economic analysis and policy-making.