FORECASTING THE INFLATION RATES IN KENYA

KIBUNYWA SAMUEL

2025-07-21

```r
library(readxl)
```

```
## Warning: package 'readxl' was built under R version 4.4.3
```

```r
Inflation_dataset <- read_excel("C:/Users/Wintham/Desktop/Sir_ME/Inflation data(1).xlsx")
```

```
## New names:
## * `Month` -> `Month...3`
## * `Month` -> `Month...4`
## * `` -> `...6`
```

```r
head(Inflation_dataset)
```

```
## # A tibble: 6 x 7
##    Year   Day Month...3 Month...4 `12-Month Inflation` ...6  `Imported date`
##   <dbl> <dbl> <chr>         <dbl>                <dbl> <lgl> <dttm>
## 1  2005     1 December         12                 4.7  NA    2005-12-01 00:00:00
## 2  2005     1 November         11                 4.4  NA    2005-11-01 00:00:00
## 3  2005     1 October          10                 3.72 NA    2005-10-01 00:00:00
## 4  2005     1 September         9                 4.27 NA    2005-09-01 00:00:00
## 5  2005     1 August            8                 6.87 NA    2005-08-01 00:00:00
## 6  2005     1 July              7                11.8  NA    2005-07-01 00:00:00
```

1) DATA PRE-PROCESSING

```r
colnames(Inflation_dataset)
```

```
## [1] "Year"                "Day"                 "Month...3"
## [4] "Month...4"           "12-Month Inflation" "...6"
## [7] "Imported date"
```

```r
colSums(is.na(Inflation_dataset))
```

```
##               Year                Day          Month...3          Month...4
##                  0                  0                  0                  0
## 12-Month Inflation               ...6      Imported date
##                  0                246                  0
```

```r
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 4.4.3

##
## Attaching package: 'dplyr'

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

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

```r
# Renaming necessary columns
renamed <- Inflation_dataset %>%
  rename(
    inflation_rate = `12-Month Inflation`,
    date = `Imported date`
  )

# Identifying and dropping columns full of NAs and other columns
cleaned_data <- renamed %>%
  select(where(~ !all(is.na(.)))) %>%      # Removes columns that are ALL NA
  select(c(inflation_rate, date)) %>%
  mutate(date = as.Date(date, origin = "1899-12-30")) %>%         #formatting the date
  arrange(date)# Sort in ascending order


head(cleaned_data)
```

```
## # A tibble: 6 x 2
##   inflation_rate date
##            <dbl> <date>
## 1           14.9 2005-01-01
## 2           13.9 2005-02-01
## 3           14.2 2005-03-01
## 4           16.0 2005-04-01
## 5           14.8 2005-05-01
## 6           11.9 2005-06-01
```

```r
# Checking and removing duplicate rows

if (any(duplicated(cleaned_data))) {
  cat("Duplicate rows detected. Cleaning them up...\n")
  clean_data <- cleaned_data[!duplicated(cleaned_data), ]

} else {
  cat("No duplicate rows found.\n")
  clean_data <- cleaned_data
}
```

```
## Duplicate rows detected. Cleaning them up...
```

```r
train_data <- clean_data %>%
  filter(date <= as.Date("2024-12-01"))

tail(train_data)
```

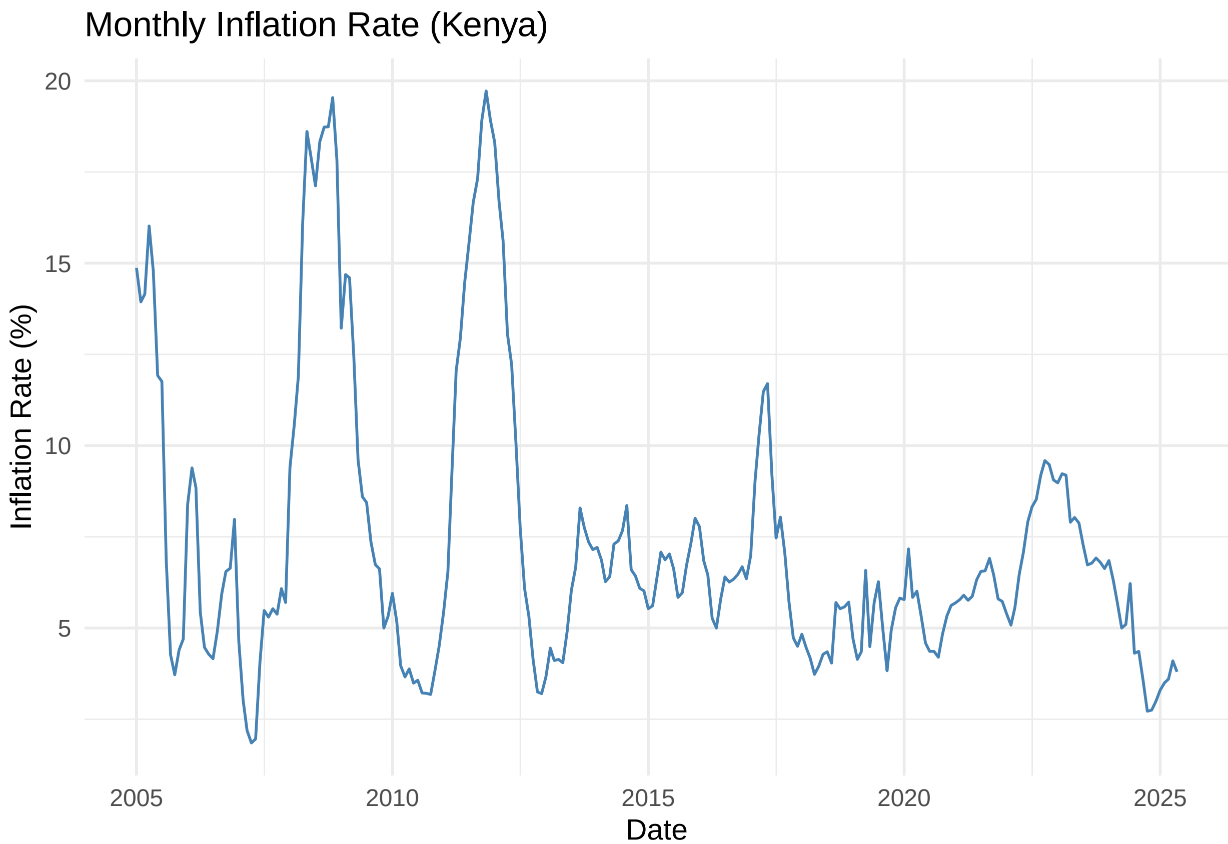
```
## # A tibble: 6 x 2
##   inflation_rate date
##            <dbl> <date>
## 1           4.31 2024-07-01
## 2           4.36 2024-08-01
## 3           3.56 2024-09-01
## 4           2.72 2024-10-01
## 5           2.75 2024-11-01
## 6           2.99 2024-12-01
```

2) EXPLORATORY DATA ANALYSIS(EDA)

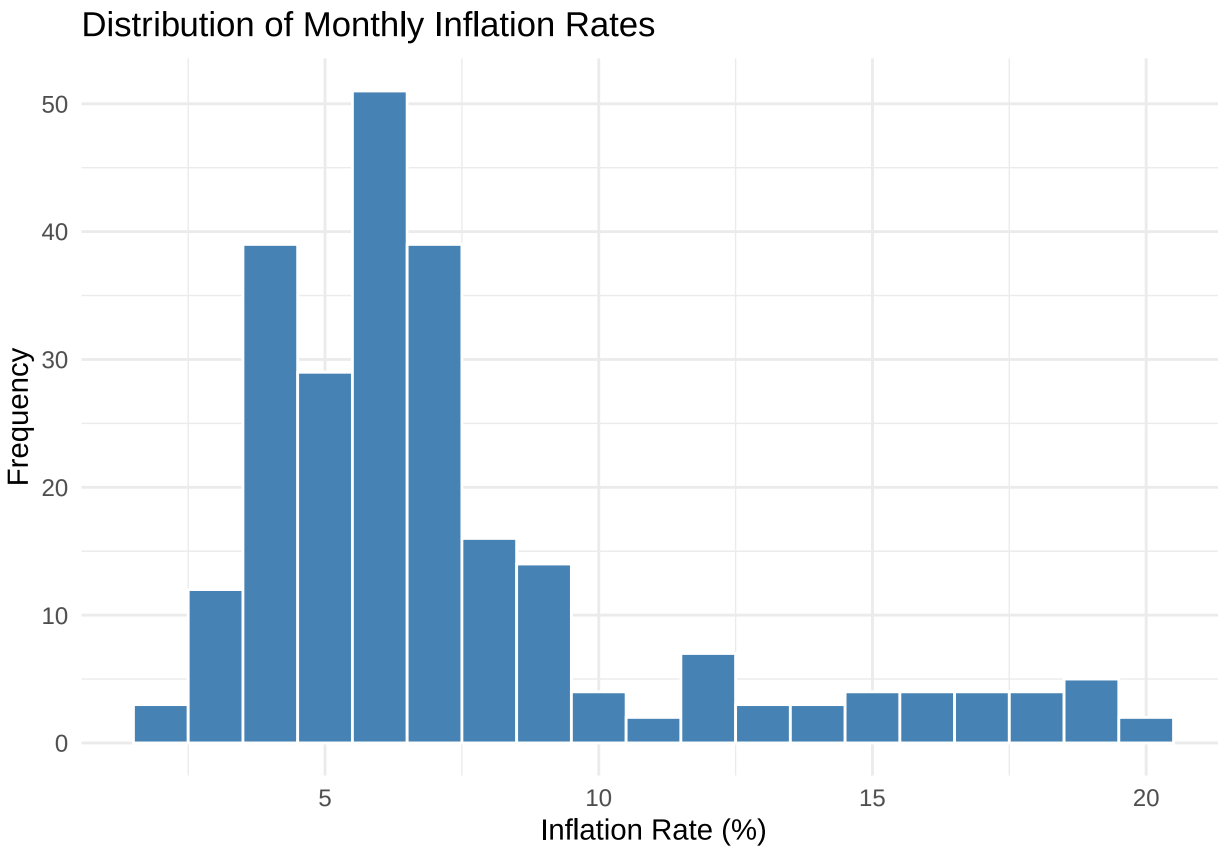
```r
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 4.4.3
```

```r
ggplot(clean_data, aes(x = date, y = inflation_rate)) +
  geom_line(color = "steelblue") +
  labs(title = "Monthly Inflation Rate (Kenya)",
       x = "Date",
       y = "Inflation Rate (%)") +
  theme_minimal()
```

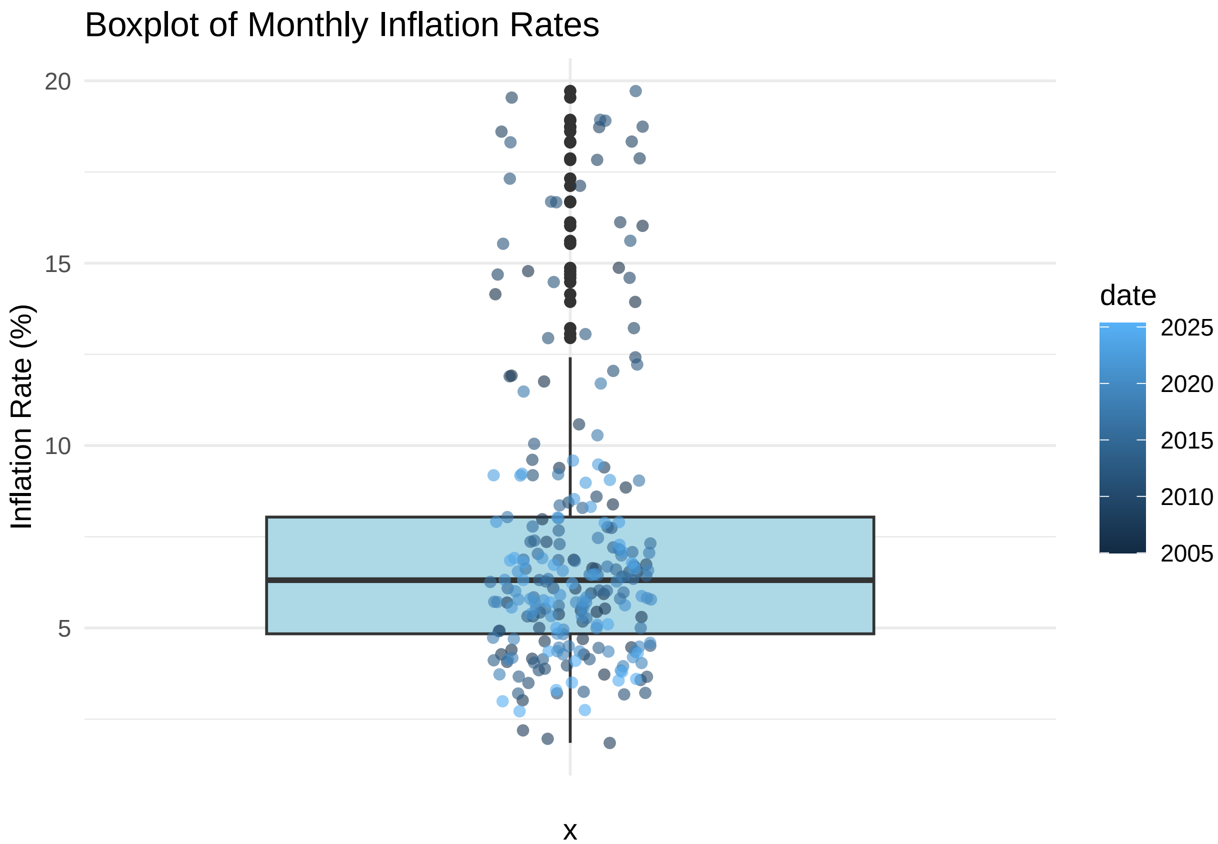



```r
ggplot(clean_data, aes(x = inflation_rate)) +
  geom_histogram(binwidth = 1, fill = "steelblue", color = "white") +
  labs(title = "Distribution of Monthly Inflation Rates",
       x = "Inflation Rate (%)",
       y = "Frequency") +
  theme_minimal()
```




```r
ggplot(clean_data, aes(y = inflation_rate)) +
  geom_boxplot(fill = "steelblue", color = "black") +
  labs(title = "Boxplot of Monthly Inflation Rates",
       y = "Inflation Rate (%)") +
  theme_minimal()
```


```r
ggplot(clean_data, aes(x = "", y = inflation_rate)) +
  geom_boxplot(fill = "lightblue") +
  geom_jitter(aes(color = date), width = 0.1, alpha = 0.6) +
  labs(title = "Boxplot of Monthly Inflation Rates",
       y = "Inflation Rate (%)") +
  theme_minimal()
```




```
# Calculate IQR bounds
Q1 <- quantile(clean_data$inflation_rate, 0.25)
Q3 <- quantile(clean_data$inflation_rate, 0.75)
IQR_val <- Q3 - Q1

lower_bound <- Q1 - 1.5 * IQR_val
upper_bound <- Q3 + 1.5 * IQR_val

# Filter and label outliers with direction
outlier_months <- clean_data %>%
  filter(inflation_rate < lower_bound | inflation_rate > upper_bound) %>%
  mutate(outlier_type = ifelse(inflation_rate < lower_bound, "Low", "High"))

print(outlier_months)
```

```
## # A tibble: 29 x 3
##    inflation_rate date       outlier_type
##             <dbl> <date>     <chr>
##  1           14.9 2005-01-01 High
##  2           13.9 2005-02-01 High
##  3           14.2 2005-03-01 High
##  4           16.0 2005-04-01 High
##  5           14.8 2005-05-01 High
##  6           16.1 2008-04-01 High
##  7           18.6 2008-05-01 High
##  8           17.9 2008-06-01 High
##  9           17.1 2008-07-01 High
```

```
## 10           18.3 2008-08-01 High
## # i 19 more rows
```

I decided to keep the outliers because I assumed they represent meaningful economic events: such as global oil prices spiked, or local taxes shifted

For example, in year 2008, 9 of 12 months had very high inflation rates

This might have been a result of the 2007 post-election violence

AND 2011 rates might have been influenced by the 2010 constitution ammendment.

It is also good to note all the outliers were all high

```r
train_data$outlier_dummy <- ifelse(train_data$date %in% outlier_months$date, 1, 0)
head(train_data)
```

```
## # A tibble: 6 x 3
##   inflation_rate date       outlier_dummy
##            <dbl> <date>             <dbl>
## 1           14.9 2005-01-01             1
## 2           13.9 2005-02-01             1
## 3           14.2 2005-03-01             1
## 4           16.0 2005-04-01             1
## 5           14.8 2005-05-01             1
## 6           11.9 2005-06-01             0
```

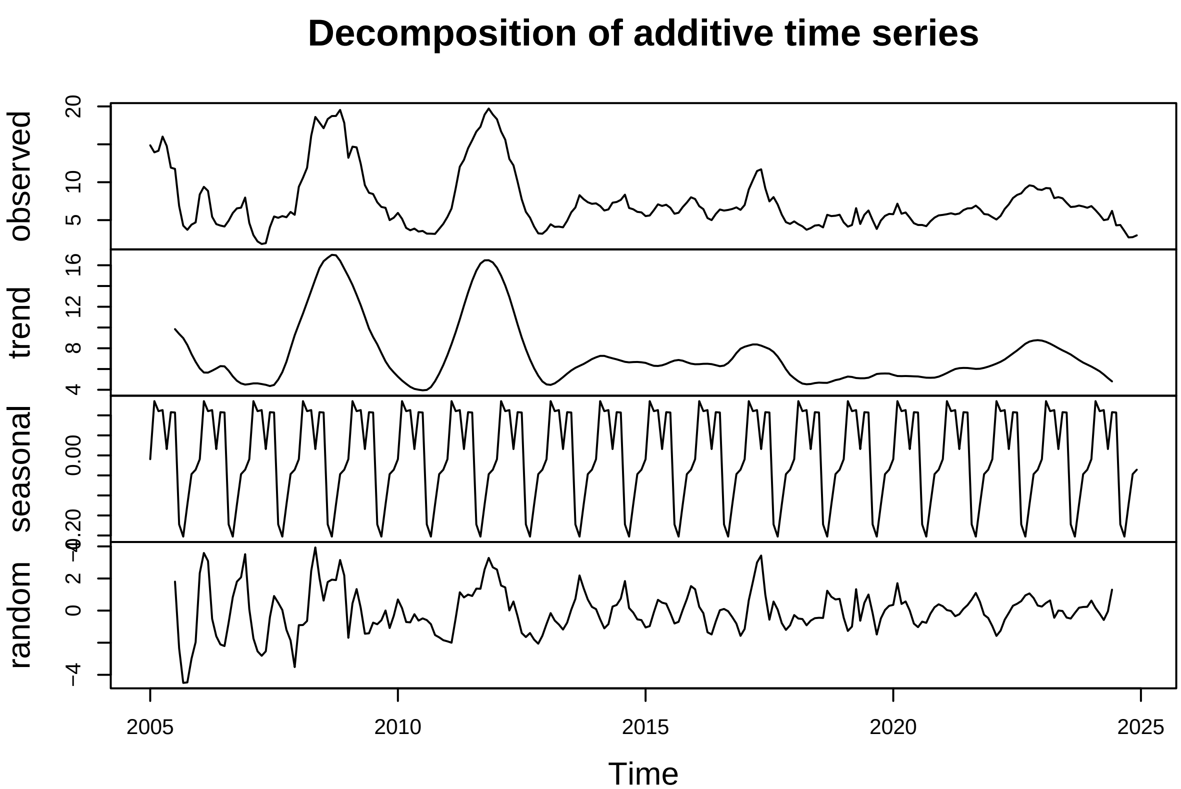
3) MODEL BUILDING

```r
ts_data <- ts(train_data$inflation_rate, start = c(2005, 1), frequency = 12)
ts_data
```

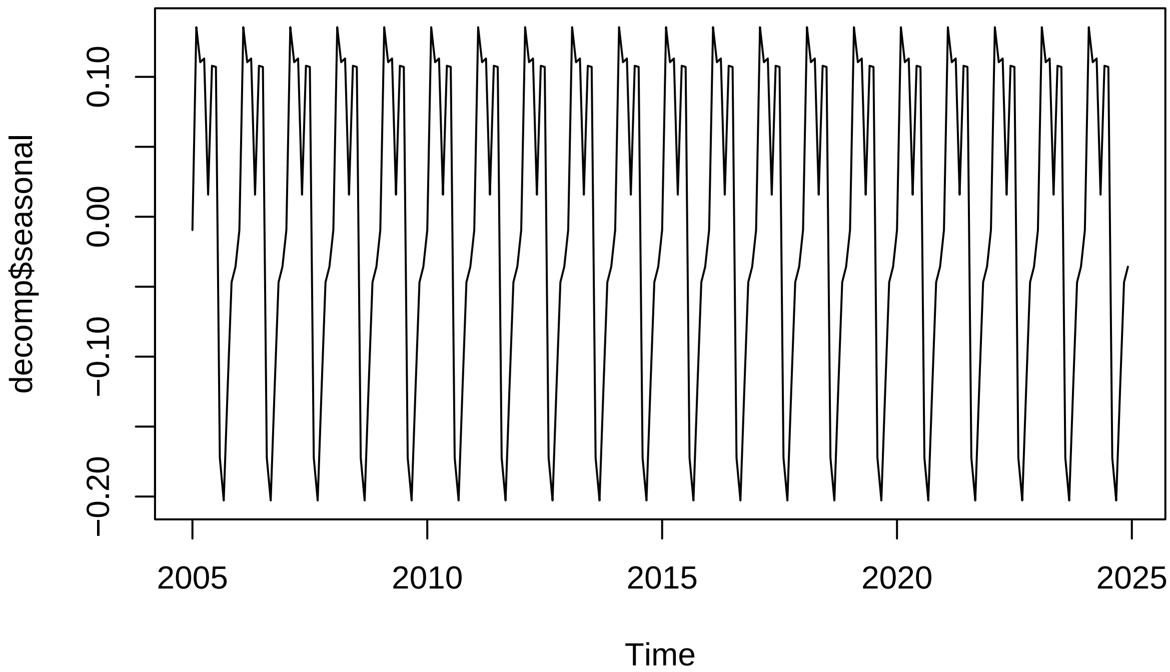
```
##        Jan   Feb   Mar   Apr   May   Jun   Jul   Aug   Sep   Oct   Nov   Dec
## 2005 14.87 13.94 14.15 16.02 14.78 11.92 11.76  6.87  4.27  3.72  4.40  4.70
## 2006  8.39  9.39  8.85  5.44  4.47  4.28  4.16  4.92  5.93  6.55  6.64  7.98
## 2007  4.63  3.02  2.19  1.85  1.96  4.07  5.48  5.30  5.53  5.38  6.08  5.70
## 2008  9.40 10.58 11.90 16.12 18.61 17.87 17.12 18.33 18.73 18.74 19.54 17.83
## 2009 13.22 14.69 14.60 12.42  9.61  8.60  8.44  7.36  6.74  6.62  5.00  5.32
## 2010  5.95  5.18  3.97  3.66  3.88  3.49  3.57  3.22  3.21  3.18  3.84  4.51
```

```
## 2011  5.42  6.54  9.19 12.05 12.95 14.48 15.53 16.67 17.32 18.91 19.72 18.93
## 2012 18.31 16.69 15.61 13.06 12.22 10.05  7.74  6.09  5.32  4.14  3.25  3.20
## 2013  3.67  4.45  4.11  4.14  4.05  4.91  6.03  6.67  8.29  7.76  7.36  7.15
## 2014  7.21  6.86  6.27  6.41  7.30  7.39  7.67  8.36  6.60  6.43  6.09  6.02
## 2015  5.53  5.61  6.31  7.08  6.87  7.03  6.62  5.84  5.97  6.72  7.32  8.01
## 2016  7.78  6.84  6.45  5.27  5.00  5.80  6.40  6.26  6.34  6.47  6.68  6.35
## 2017  6.99  9.04 10.28 11.48 11.70  9.21  7.47  8.04  7.06  5.72  4.73  4.50
## 2018  4.83  4.46  4.18  3.73  3.95  4.28  4.35  4.04  5.70  5.53  5.58  5.71
## 2019  4.70  4.14  4.35  6.58  4.49  5.70  6.27  5.00  3.83  4.95  5.56  5.82
## 2020  5.78  7.17  5.84  6.01  5.33  4.59  4.36  4.36  4.20  4.84  5.33  5.62
## 2021  5.69  5.78  5.90  5.76  5.87  6.32  6.55  6.57  6.91  6.45  5.80  5.73
## 2022  5.39  5.08  5.56  6.47  7.08  7.91  8.32  8.53  9.18  9.59  9.48  9.06
## 2023  8.98  9.23  9.19  7.90  8.03  7.88  7.28  6.73  6.78  6.92  6.80  6.63
## 2024  6.85  6.31  5.70  5.00  5.10  6.22  4.31  4.36  3.56  2.72  2.75  2.99
```

```r
# Plot all components
decomp <- decompose(ts_data)
plot(decomp)
```




```r
plot(decomp$seasonal)
```



LOG TRANSFORMATION

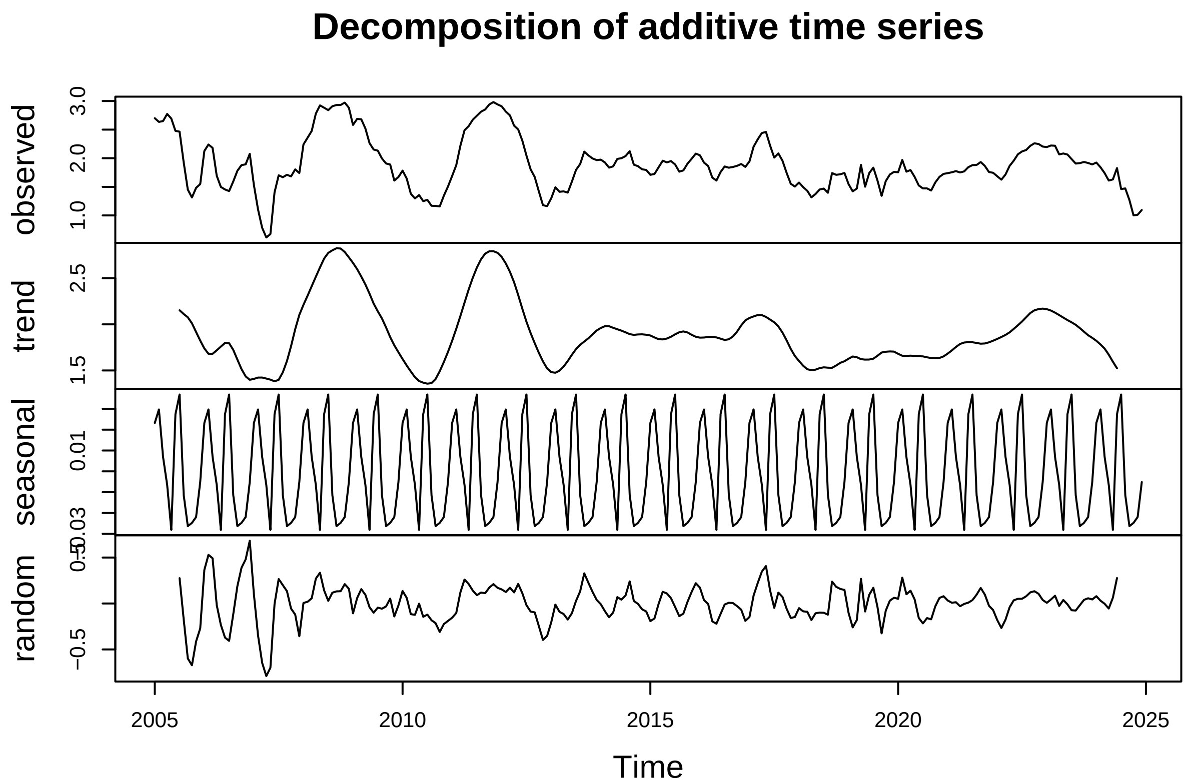
Stabilizes variance: When data has bigger swings as values increase, logging tames the volatility.

Linearizes exponential growth: If inflation is rising faster and faster, logs make it appear more linear.

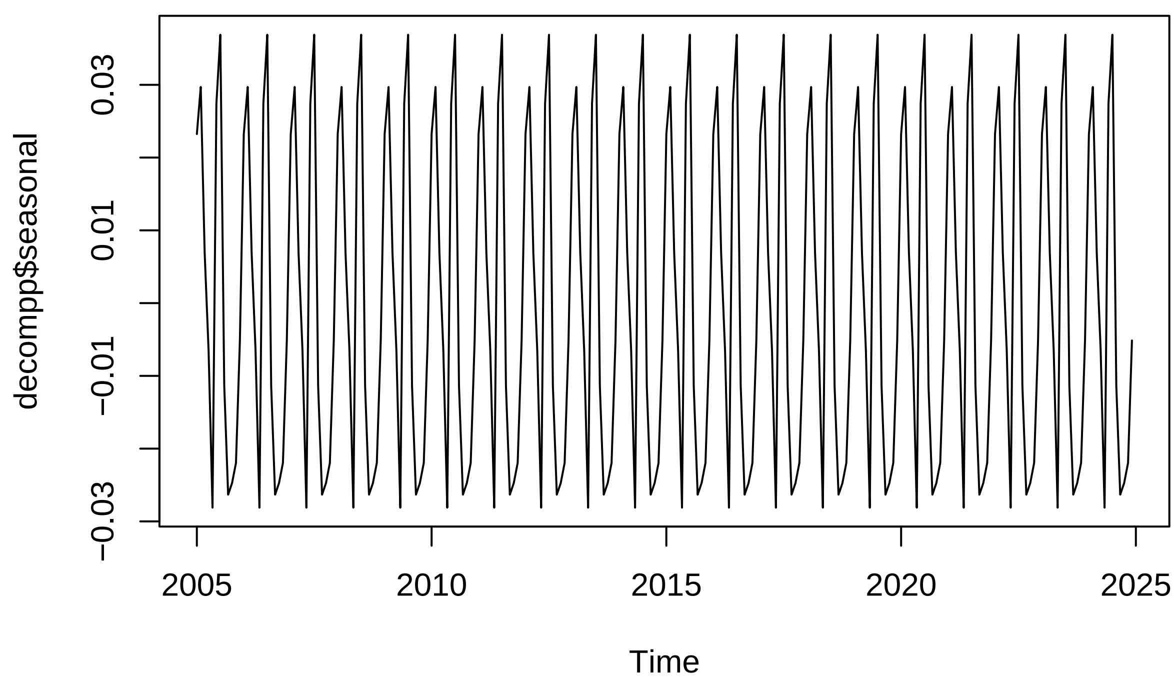
Turns multiplicative seasonality into additive: This helps methods like decomposition or SARIMA, which assume additive components.

when inflation rates rise rapidly, they tend to have multiplicative seasonality

```
log_ts <- log(ts_data)
decompp <- decompose(log_ts)
plot(decompp)
```




```r
plot(decompp$seasonal)
```




```r
#install.packages("tseries")

library(tseries)
```

```
## Warning: package 'tseries' was built under R version 4.4.3
```

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

```r
adf.test(log_ts)
```

```
## Warning in adf.test(log_ts): p-value smaller than printed p-value
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  log_ts
## Dickey-Fuller = -4.2003, Lag order = 6, p-value = 0.01
## alternative hypothesis: stationary
```

```r
cat("------------------------------------")
```

```
## ------------------------------------
```

```r
library(urca)
```

```
## Warning: package 'urca' was built under R version 4.4.3
```

```r
kpss_test <- ur.kpss(log_ts, type = "tau")
summary(kpss_test)
```

```
##
## #######################
## # KPSS Unit Root Test #
## #######################
##
## Test is of type: tau with 4 lags.
##
## Value of test-statistic is: 0.0527
##
## Critical value for a significance level of:
##                 10pct  5pct 2.5pct  1pct
## critical values 0.119 0.146  0.176 0.216
```

Te p-value (0.01) is less than 0.05

We reject the null hypothesis

Our data is stationary and no differencing is required.

KPSS confirms that it's not trend-stationary, it's just outright stationary

All critical values (even 10%) are above the test statistic

Combined, both tests gives a high confidence that no differencing is needed (d = 0)

```r
#install.packages('forecast')

library(forecast)
```

```
## Warning: package 'forecast' was built under R version 4.4.3
```

```r
findfrequency(ts_data)
```

```
## [1] 33
```

```r
findfrequency(log_ts)
```

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

33 indicates repeated patterns are very strong at 33 months intervals.

This indicates presence of cyclic patterns in our data [good for time series]

Cycles often contain real signal that ARIMA or SARIMA models are meant to capture through autoregressive or moving average terms.

If a cycle isn't perfectly regular like seasonality, differencing may weaken it, even though it won't erase it completely.

This sets our seasonal differencing D to zero

```
# install.packages("uroot")

#ndiffs(ts_data)     # should be 0

nsdiffs(log_ts)    # may return 1 if strong seasonality
```

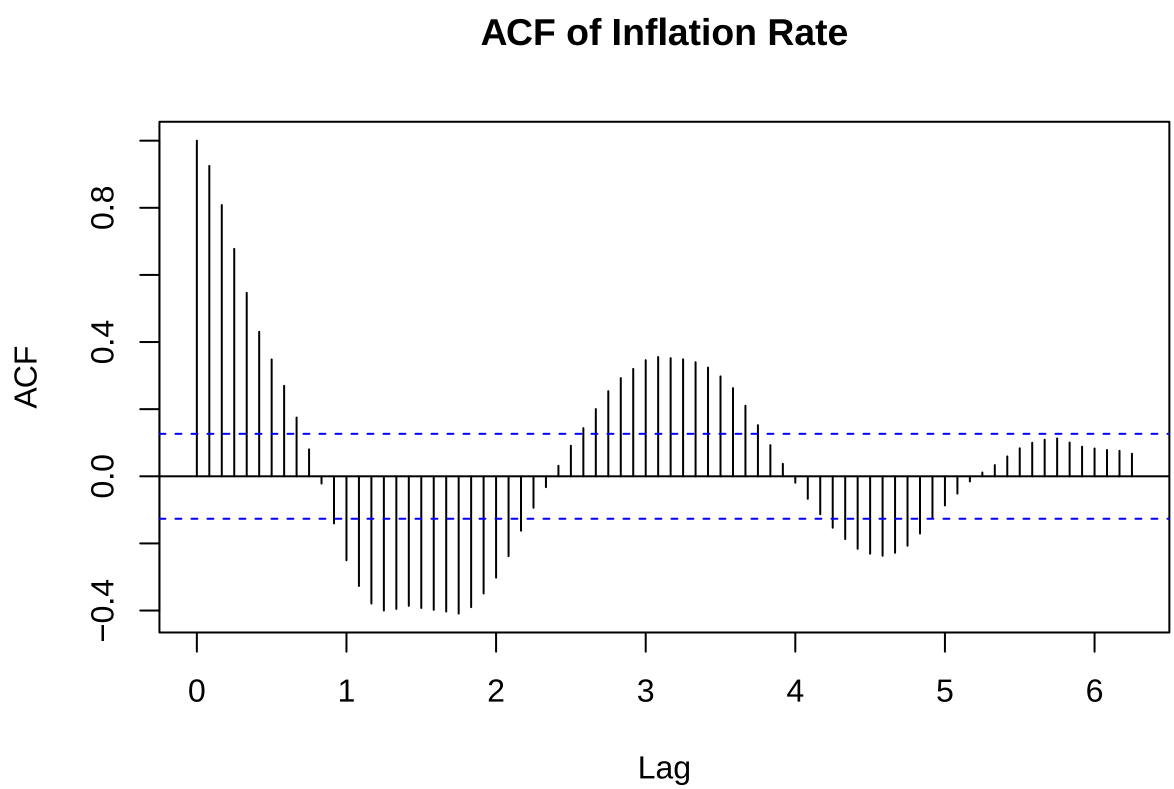
```
## [1] 0
```

```
nsdiffs(log_ts, test = "ch")
```

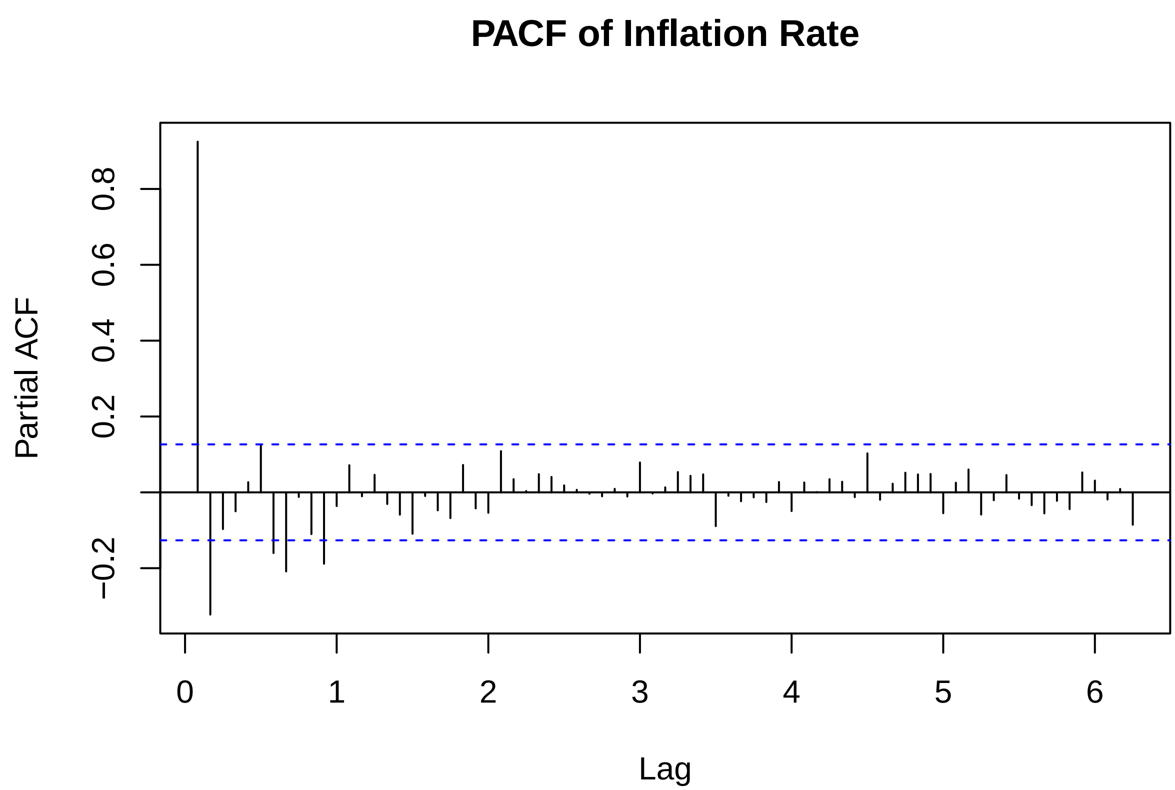
```
## [1] 0
```

The zero values indicate that NO seasonal differencing is necessary

```
acf(log_ts, lag.max = 75, main = "ACF of Inflation Rate")
```




```
pacf(log_ts, lag.max = 75, main = "PACF of Inflation Rate")
```



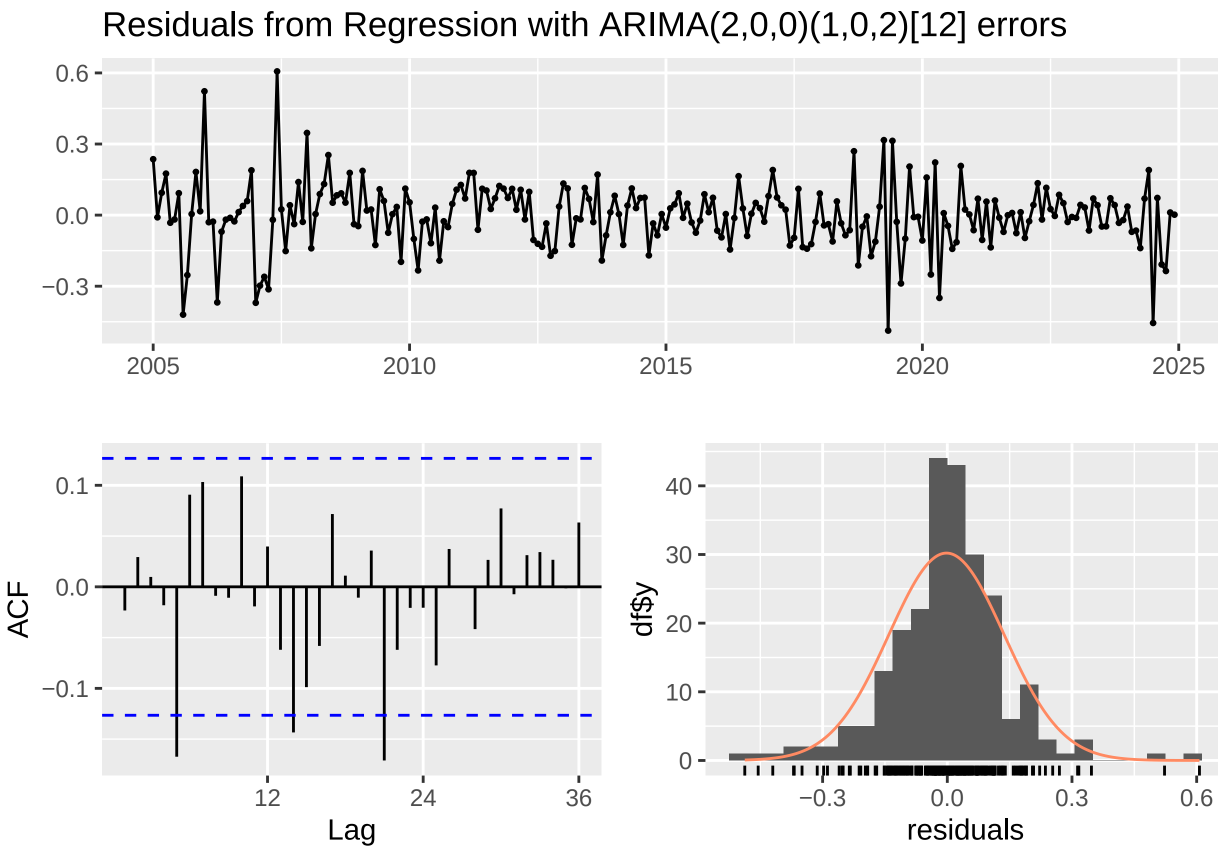
```r
manual_fit <- Arima(log_ts, order = c(2,0,0),
             seasonal = list(order = c(1,0,2), period = 12),
             xreg = train_data$outlier_dummy)

summary(manual_fit)
```

```
## Series: log_ts
## Regression with ARIMA(2,0,0)(1,0,2)[12] errors
##
## Coefficients:
```

```
## Warning in sqrt(diag(x$var.coef)): NaNs produced
```

```
##          ar1      ar2    sar1    sma1     sma2  intercept    xreg
##       1.2355  -0.3146  -0.611  0.1741  -0.2654     1.8869  0.0888
## s.e.  0.0632   0.0633     NaN     NaN      NaN     0.0649  0.0772
##
## sigma^2 = 0.01974:  log likelihood = 131.67
## AIC=-247.33   AICc=-246.71   BIC=-219.49
##
## Training set error measures:
##                        ME      RMSE        MAE        MPE     MAPE      MASE
## Training set -0.001916247 0.1384361 0.09811011 -0.8956994 5.769298 0.1800187
##                     ACF1
## Training set -0.02330015
```

```r
checkresiduals(manual_fit, lag = 48)
```



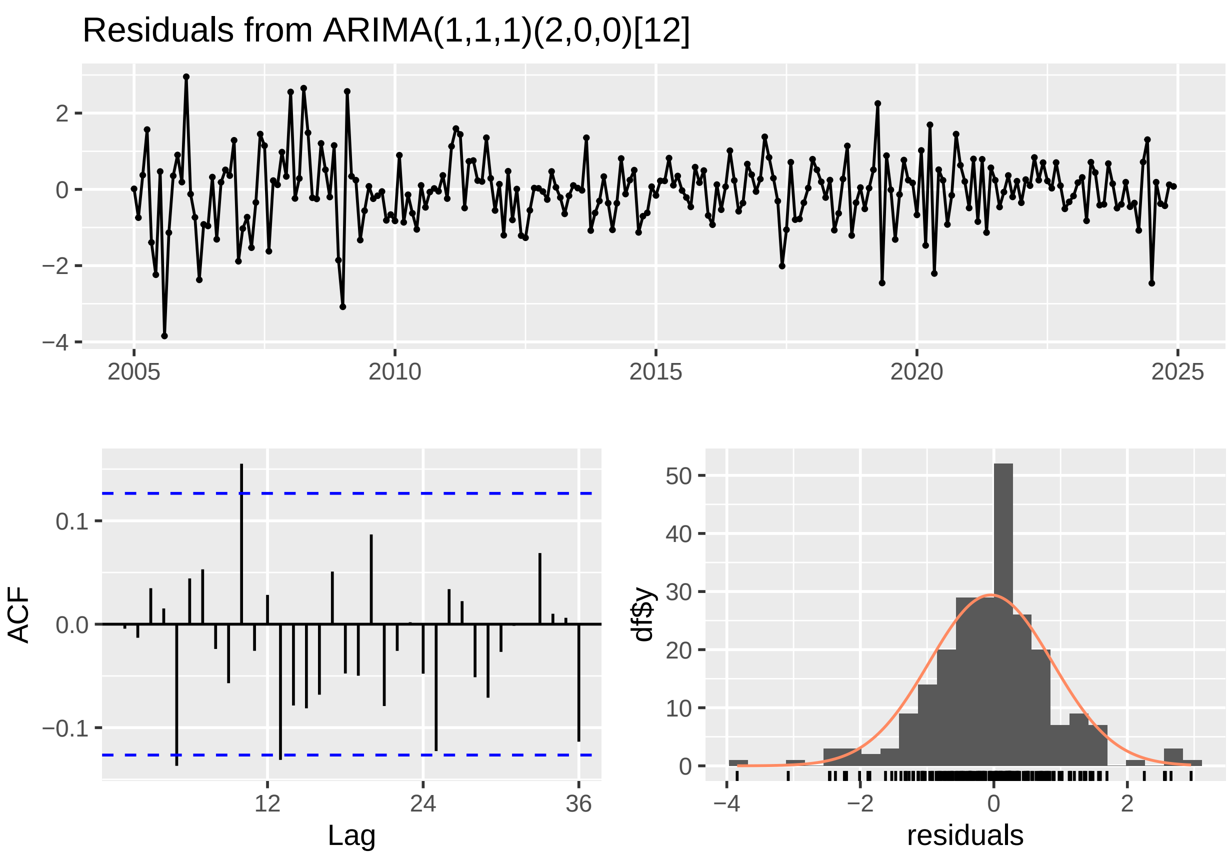

```
##
##  Ljung-Box test
##
## data:  Residuals from Regression with ARIMA(2,0,0)(1,0,2)[12] errors
## Q* = 48.827, df = 43, p-value = 0.2504
##
## Model df: 5.   Total lags used: 48
```

```
library(forecast)

auto_fit <- auto.arima(ts_data)

summary(auto_fit)
```

```
## Series: ts_data
## ARIMA(1,1,1)(2,0,0)[12]
##
## Coefficients:
##          ar1      ma1     sar1     sar2
##       0.5399  -0.3028  -0.6369  -0.3235
## s.e.  0.1904   0.2154   0.0680   0.0705
##
## sigma^2 = 0.8675:  log likelihood = -323.07
## AIC=656.14   AICc=656.39   BIC=673.52
##
## Training set error measures:
```

```
##                          ME      RMSE       MAE       MPE     MAPE     MASE
## Training set -0.04543918 0.9216304 0.6679176 -2.006202 10.48682 0.158355
##                     ACF1
## Training set -0.004320652
```

```r
checkresiduals(auto_fit, lag = 48)
```




```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(1,1,1)(2,0,0)[12]
## Q* = 49.559, df = 44, p-value = 0.2612
##
## Model df: 4.   Total lags used: 48
```

Both the manually fitted and the auto fitted models exhibit residual ACF plots that cut off immediately, indicating that non-seasonal autocorrelation has been effectively addressed

In addition, histograms with overlaid normal density curves for both models show that the residuals are approximately symmetric and bell-shaped, supporting the assumption of normality.

However, despite visual similarities, AIC and BIC values are substantially lower for the manually fitted (manual_fit), indicating a model that balances complexity and goodness-of-fit more effectively.

Lower values mean the model explains the data well while remaining efficient.

Our manually fitted model seems to be the best choice

```r
Box.test(residuals(manual_fit), type = "Ljung-Box")
```

```
##
##  Box-Ljung test
##
## data:  residuals(manual_fit)
## X-squared = 0.13193, df = 1, p-value = 0.7164
```

The null hypothesis of the Box-Ljung test is that there is no autocorrelation in the residuals at lag 1.

A high p-value (0.7299) suggests we do not reject the null hypothesis.

so: There's no significant autocorrelation in our residuals, indicating that our model residuals behave like white noise,

```r
# Forecasting next 12 months
library(forecast)

future_dummy <- rep(0, 12)
```

```r
inflation_forecast <- forecast(manual_fit, xreg = future_dummy, h = 12)
inflation_forecast
```

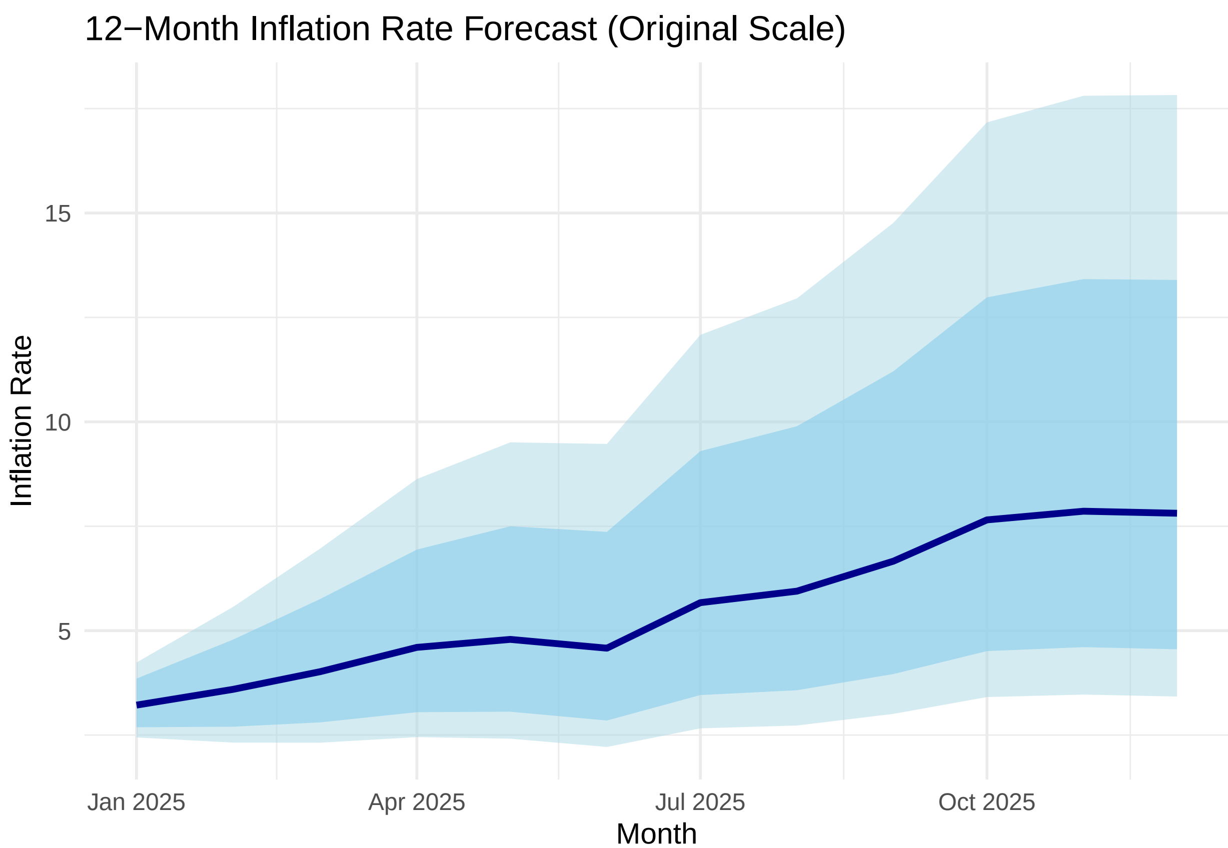
```
##          Point Forecast      Lo 80    Hi 80     Lo 95    Hi 95
## Jan 2025       1.168557 0.9884983 1.348615 0.8931811 1.443932
## Feb 2025       1.279360 0.9931653 1.565554 0.8416631 1.717056
## Mar 2025       1.390905 1.0310286 1.750782 0.8405215 1.941289
## Apr 2025       1.525895 1.1143879 1.937401 0.8965495 2.155240
## May 2025       1.566574 1.1182557 2.014893 0.8809302 2.252218
## Jun 2025       1.521727 1.0467093 1.996745 0.7952500 2.248205
## Jul 2025       1.735265 1.2406062 2.229924 0.9787495 2.491781
## Aug 2025       1.782670 1.2734029 2.291937 1.0038132 2.561527
## Sep 2025       1.896827 1.3766055 2.417048 1.1012170 2.692437
## Oct 2025       2.034992 1.5065057 2.563478 1.2267421 2.843241
## Nov 2025       2.061786 1.5270363 2.596535 1.2439569 2.879615
## Dec 2025       2.055543 1.5160310 2.595056 1.2304303 2.880657
```

```r
forecast_df <- as.data.frame(inflation_forecast)


forecast_df_transformed <- data.frame(
  Month = seq(from = as.Date("2025-01-01"), by = "month", length.out = 12),
  Point_Forecast = exp(forecast_df$`Point Forecast`),
  Lo_80 = exp(forecast_df$`Lo 80`),
  Hi_80 = exp(forecast_df$`Hi 80`),
  Lo_95 = exp(forecast_df$`Lo 95`),
  Hi_95 = exp(forecast_df$`Hi 95`)
)


print(forecast_df_transformed, row.names = FALSE)
```

```
##       Month Point_Forecast    Lo_80     Hi_80    Lo_95     Hi_95
##  2025-01-01       3.217345 2.687196  3.852086 2.442888  4.237324
##  2025-02-01       3.594338 2.699767  4.785326 2.320223  5.568114
##  2025-03-01       4.018486 2.803948  5.759103 2.317575  6.967725
##  2025-04-01       4.599256 3.047702  6.940691 2.451131  8.629959
##  2025-05-01       4.790210 3.059513  7.499924 2.413143  9.508807
##  2025-06-01       4.580129 2.848263  7.365045 2.214995  9.470716
##  2025-07-01       5.670432 3.457709  9.299163 2.661126 12.082778
##  2025-08-01       5.945711 3.572991  9.894086 2.728667 12.955585
##  2025-09-01       6.664713 3.961432 11.212714 3.007824 14.767619
##  2025-10-01       7.652189 4.510941 12.980883 3.410102 17.171333
##  2025-11-01       7.859994 4.604510 13.417172 3.469314 17.807410
##  2025-12-01       7.811082 4.554114 13.397337 3.422702 17.825974
```

```r
library(ggplot2)

ggplot(forecast_df_transformed, aes(x = Month)) +
  geom_ribbon(aes(ymin = Lo_95, ymax = Hi_95), fill = "lightblue", alpha = 0.5) +
  geom_ribbon(aes(ymin = Lo_80, ymax = Hi_80), fill = "skyblue", alpha = 0.6) +
  geom_line(aes(y = Point_Forecast), color = "blue4", linewidth = 1.2) +
```

```r
  labs(
    title = "12-Month Inflation Rate Forecast (Original Scale)",
    x = "Month",
    y = "Inflation Rate"
  ) +
  theme_minimal()
```



4) MODEL EVALUATION

```r
test_data <- clean_data %>%
  filter(date >= as.Date("2025-01-01"))

head(test_data)
```

```
## # A tibble: 5 x 2
##   inflation_rate date
##            <dbl> <date>
## 1            3.3 2025-01-01
## 2            3.5 2025-02-01
## 3            3.6 2025-03-01
## 4            4.1 2025-04-01
## 5            3.8 2025-05-01
```

```r
test_2025_clean <- test_data %>%
  rename(actual = `inflation_rate`)

comparison_df <- forecast_df_transformed %>%
  select(date = Month, forecast = Point_Forecast, Lo_95, Hi_95) %>%
  inner_join(test_2025_clean, by = "date")

head(comparison_df)
```

```
##         date forecast    Lo_95    Hi_95 actual
## 1 2025-01-01 3.217345 2.442888 4.237324    3.3
## 2 2025-02-01 3.594338 2.320223 5.568114    3.5
## 3 2025-03-01 4.018486 2.317575 6.967725    3.6
## 4 2025-04-01 4.599256 2.451131 8.629959    4.1
## 5 2025-05-01 4.790210 2.413143 9.508807    3.8
```

```r
#install.packages("Metrics")
library(Metrics)
```

```
## Warning: package 'Metrics' was built under R version 4.4.3

##
## Attaching package: 'Metrics'

## The following object is masked from 'package:forecast':
##
##     accuracy
```

```r
rmse_val <- rmse(comparison_df$actual, comparison_df$forecast)
mae_val  <- mae(comparison_df$actual, comparison_df$forecast)
mape_val <- mape(comparison_df$actual, comparison_df$forecast) * 100

cat(
  " Model Performance Summary\n",
  "--------------\n",
  " RMSE  = ", round(rmse_val, 2),
  "\n The model's average prediction error is ±", round(rmse_val, 2), " percentage points.\n",
  "\n MAE   = ", round(mae_val, 2),
  "\n On average, forecasts deviate by ", round(mae_val, 2), " points from actual inflation.\n",
  "\n MAPE  = ", round(mape_val, 2), "%",
  " \n Forecasts are within ", 100 - round(mape_val, 2), "% of actual values, on average.\n"
)
```

```
##  Model Performance Summary
##  --------------
##   RMSE  =  0.53
##  The model's average prediction error is ± 0.53  percentage points.
##
##  MAE   =  0.42
##  On average, forecasts deviate by  0.42  points from actual inflation.
##
##  MAPE  =  11.01 %
##  Forecasts are within  88.99 % of actual values, on average.
```

```
within_interval <- with(comparison_df, actual >= Lo_95 & actual <= Hi_95)
coverage_rate <- mean(within_interval) * 100

cat("Interval coverage rate:", round(coverage_rate, 1), "%\n")
```

```
## Interval coverage rate: 100 %
```

Interval coverage rate: 100 %

The forecasted values completely lies within the 95% confidence intervals

THE MODEL PERFORMED EXCELLENTLY!!

5) MODEL DEPLOYMENT

```
forecasted <- forecast_df_transformed%>%
  filter(Month > as.Date("2025-05-01"))
forecasted
```

```
##        Month Point_Forecast    Lo_80     Hi_80    Lo_95     Hi_95
## 1 2025-06-01       4.580129 2.848263  7.365045 2.214995  9.470716
## 2 2025-07-01       5.670432 3.457709  9.299163 2.661126 12.082778
## 3 2025-08-01       5.945711 3.572991  9.894086 2.728667 12.955585
## 4 2025-09-01       6.664713 3.961432 11.212714 3.007824 14.767619
## 5 2025-10-01       7.652189 4.510941 12.980883 3.410102 17.171333
## 6 2025-11-01       7.859994 4.604510 13.417172 3.469314 17.807410
## 7 2025-12-01       7.811082 4.554114 13.397337 3.422702 17.825974
```

Between June and December 2025, Kenya's inflation rate is forecasted to follow an upward trend, rising from 4.58% in June to a peak of approximately 7.86% in November.

The widening confidence intervals in later months—particularly the 95% bounds reaching up to 17.83%—signal increasing uncertainty due to long-term volatility or potential shocks.

For instance, the July forecast suggests an inflation rate of 5.67%, but with a 95% confidence range spanning from 2.66% to 12.08%, underscoring the importance of cautious policy interpretation and scenario planning