

ARIMA model in R

Frank Zhong

2024-10-04

ARIMA model in R

The ARIMA model (Auto Regressive Integrated Moving Average) is a popular time series forecasting method used to predict future values based on past data. It's widely applied in fields like economics, finance, and environmental science for analyzing patterns and making forecasts.

The model has three main components:

AR (Auto Regressive) – This part involves regressing the variable on its own previous values. It represents the relationship between the current value and a specified number of lagged values (i.e., past observations).

I (Integrated) – The “I” component represents the differencing of raw observations to make the time series stationary, meaning it has a consistent mean and variance over time. This helps to remove trends and seasonality, making the series easier to predict.

MA (Moving Average) – The “MA” part uses past forecast errors to adjust the model. This component captures the noise or error in the data that might otherwise be unaccounted for.

The ARIMA model is typically denoted as $ARIMA(p, d, q)$, where:

p is the order of the auto regressive part, d is the degree of differencing, q is the order of the moving average part. To use ARIMA effectively, it's essential to make the time series stationary and determine the optimal values for p , d , and q , often through techniques like ACF (Autocorrelation Function) and PACF (Partial Autocorrelation Function) plots.

We first library the packages required for implementing ARIMA model in R.

```
library(readxl)
library(tidyverse)
library(lubridate)
library(ggplot2)
library(patchwork)
library(gridExtra)
library(tseries)
library(forecast)
```

For this example, we use a data of the amount of water in a lake over a certain time period. First import the data.

```
dt2 <- read_xlsx(path = "/Users/frank/Desktop/data.xlsx")
head(dt2)
```

Parameter d

There are two ways to determine whether the data need differencing or not, or furthermore, to determine the value for parameter d . We can plot the original data, first-order-differenced data, second-order-differenced data, and so on, and check for stability in data visually. Or, we can perform the ADF test for stability test numerically.

```
diff1 <- diff(dt2$amount)
diff1 <- c(diff1, NA)
dt.diff1 <- as.data.frame(diff1)
dt.diff1 <- dt.diff1 %>% mutate(date = dt2$date)
dt.diff1 <- na.omit(dt.diff1)

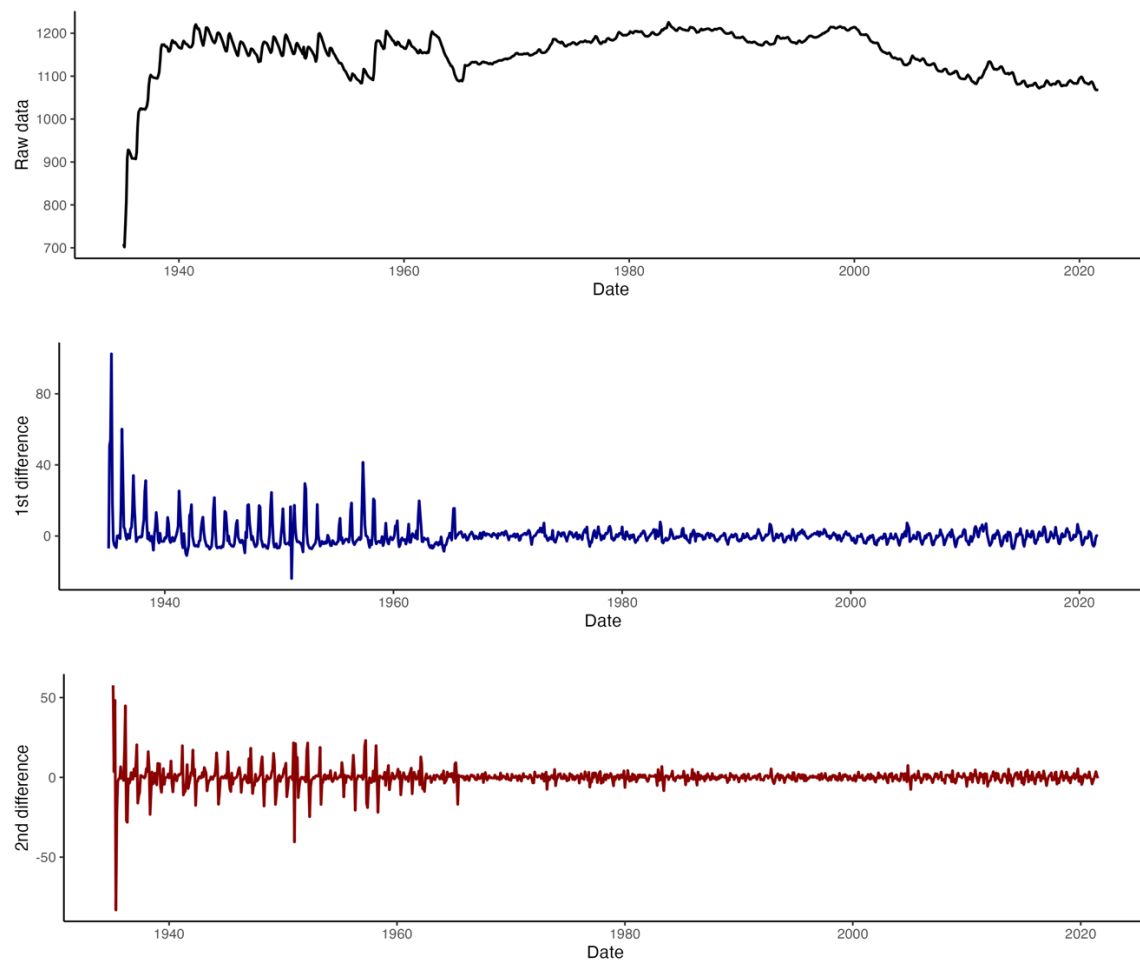
diff2 <- diff(dt.diff1$diff1)
diff2 <- c(diff2, NA)
dt.diff2 <- as.data.frame(diff2)
dt.diff2 <- dt.diff2 %>% mutate(date = dt.diff1$date)
dt.diff2 <- na.omit(dt.diff2)

p1 <- ggplot(dt2, aes(x = date, y = amount)) +
  geom_line(color = "black", linewidth = 0.8) +
  labs(title = "",
       x = "Date",
       y = "Raw data") +
  theme_classic()

p2 <- ggplot(dt.diff1, aes(x = date, y = diff1)) +
  geom_line(color = "darkblue", linewidth = 0.8) +
  labs(title = "",
       x = "Date",
       y = "1st difference") +
  theme_classic()

p3 <- ggplot(dt.diff2, aes(x = date, y = diff2)) +
  geom_line(color = "darkred", linewidth = 0.8) +
  labs(title = "",
       x = "Date",
       y = "2nd difference") +
  theme_classic()

diff.plots <- grid.arrange(p1, p2, p3, ncol = 1)
```



Then we perform the ADF tests (note that the p value need to be less than 0.05 for stable time series).

```
adf.test(na.omit(dt2$amount))

## Warning in adf.test(na.omit(dt2$amount)): p-value smaller than printed p-value

##
## Augmented Dickey-Fuller Test
##
## data: na.omit(dt2$amount)
## Dickey-Fuller = -5.3757, Lag order = 10, p-value = 0.01
## alternative hypothesis: stationary

adf.test(na.omit(dt.diff1$diff1))

## Warning in adf.test(na.omit(dt.diff1$diff1)): p-value smaller than printed
## p-value
```

```
##
## Augmented Dickey-Fuller Test
##
## data: na.omit(dt.diff1$diff1)
## Dickey-Fuller = -7.4656, Lag order = 10, p-value = 0.01
## alternative hypothesis: stationary

adf.test(na.omit(dt.diff2$diff2))

## Warning in adf.test(na.omit(dt.diff2$diff2)): p-value smaller than p
## printed
## p-value

##
## Augmented Dickey-Fuller Test
##
## data: na.omit(dt.diff2$diff2)
## Dickey-Fuller = -28.951, Lag order = 10, p-value = 0.01
## alternative hypothesis: stationary
```

Note that the test claims that the original data is considered stable, while visually it may not be clear.

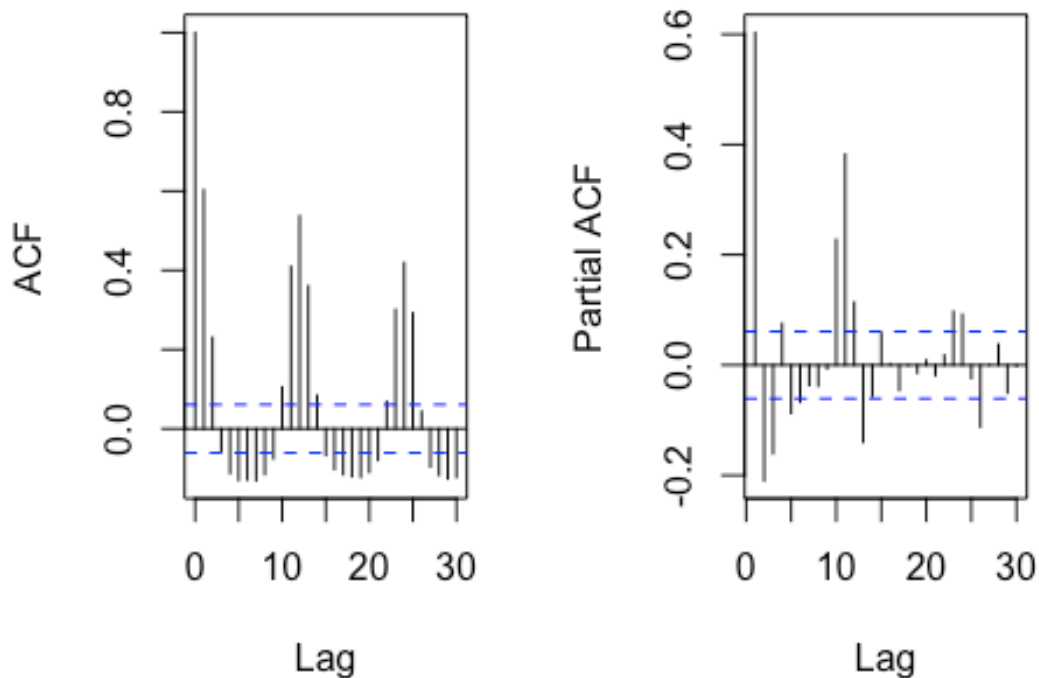
Parameters p and q

Before plotting the ACF and PACF plots, we clear the environment leaving only necessary data frames.

```
rm(list=ls()[!ls() %in% c("dt.diff1", "dt.diff2", "dt2")])
```

An easy way to plot the ACF and PACf plots is to use the `acf` and `pacf` functions.

```
par(mfrow = c(1, 2))
acf(na.omit(dt.diff1$diff1), main = "") # to determine q
pacf(na.omit(dt.diff1$diff1), main = "") # to determine p
```



However, for better appearance, we use the package `ggplot2` instead.

```
acf_values <- acf(na.omit(dt.diff1$diff1), plot = FALSE)
pacf_values <- pacf(na.omit(dt.diff1$diff1), plot = FALSE)

acf_data <- data.frame(
  Lag = acf_values$lag,
  ACF = acf_values$acf
)

pacf_data <- data.frame(
  Lag = pacf_values$lag,
  PACF = pacf_values$acf
)

acf_plot <- ggplot(acf_data, aes(x = Lag, y = ACF)) +
  geom_bar(stat = "identity", fill = "darkblue") +
  geom_hline(yintercept = 0, linetype = "dashed") +
  labs(title = "", x = "Lag", y = "ACF") +
  theme_minimal() +
  theme(
    plot.title = element_text(size = 16),      # Title size
    axis.title.x = element_text(size = 14),    # X-axis title size
  )
```

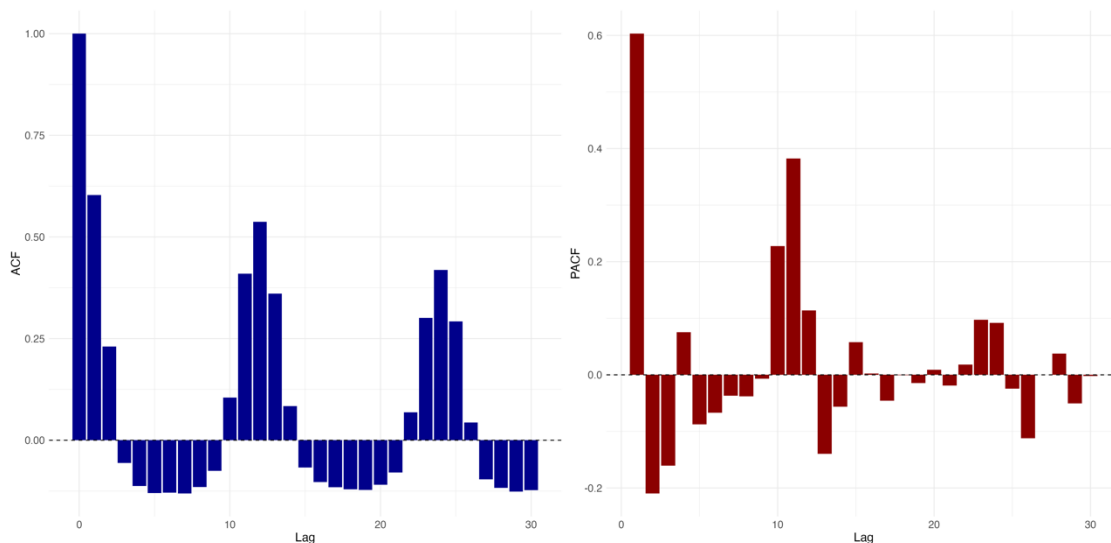
```

    axis.title.y = element_text(size = 14),      # Y-axis title size
    axis.text.x = element_text(size = 12),      # X-axis labels size
    axis.text.y = element_text(size = 12)       # Y-axis labels size
  )

pacf_plot <- ggplot(pacf_data, aes(x = Lag, y = PACF)) +
  geom_bar(stat = "identity", fill = "darkred") +
  geom_hline(yintercept = 0, linetype = "dashed") +
  labs(title = "", x = "Lag", y = "PACF") +
  theme_minimal() +
  theme(
    plot.title = element_text(size = 16),      # Title size
    axis.title.x = element_text(size = 14),    # X-axis title size
    axis.title.y = element_text(size = 14),    # Y-axis title size
    axis.text.x = element_text(size = 12),    # X-axis labels size
    axis.text.y = element_text(size = 12)     # Y-axis labels size
  )

ACF_PACF <- grid.arrange(acf_plot, pacf_plot, ncol = 2)

```



Fitting the model

Then we can fit the ARIMA model to our data.

```

water_amount_diff1 <- dt.diff1$diff1
arima.data <- ts(water_amount_diff1, start = c(1935, 2), frequency = 12)
arima.data.df <- as.data.frame(arima.data)

model <- arima(arima.data.df$x, order = c(3, 1, 3))
summary(model)

```

```
##
## Call:
## arima(x = arima.data.df$x, order = c(3, 1, 3))
##
## Coefficients:
##          ar1          ar2          ar3          ma1          ma2          ma3
##      -0.8668  -0.9942  -0.8614   0.8154   0.9830   0.7405
## s.e.    0.0687   0.0070   0.0699   0.0766   0.0216   0.0794
##
## sigma^2 estimated as 33.15:  log likelihood = -3291.25,  aic = 6596.
51
##
## Training set error measures:
##              ME      RMSE      MAE    MPE  MAPE      MASE
ACF1
## Training set 0.006686454 5.754624 3.119545 -Inf  Inf 1.101308 0.0417
5501
```

To plot the results, we define a new data frame and then draw the line graph using ggplot2.

```
fitted_values <- fitted(model)

plot_data <- data.frame(
  Date = na.omit(dt.diff1$date),
  Actual = as.numeric(dt.diff1$diff1),
  Fitted = as.numeric(fitted_values)
)
```

Then we make forecast based on the fitted ARIMA model. Note that the parameter h in the forecast function is the number of future time periods that you want to predict.

```
forecasted_values <- forecast(model, h = 20)
forecasted_values_df <- as.data.frame(forecasted_values)
forecasted_values_df <- forecasted_values_df %>% mutate(Date = rownames(
  forecasted_values_df))
date_converted <- as.Date(paste("01", forecasted_values_df$Date), forma
t="%d %b %Y")
forecasted_values_df <- forecasted_values_df %>% mutate(Date = date_con
verted)
```

Finally we plot the results.

```
last_date <- max(plot_data$Date, na.rm = TRUE)
forecasted_values_df$Date <- seq(last_date + months(1), by = "month", l
ength.out = nrow(forecasted_values_df))


ARIMA_p1 <- ggplot() +
  geom_line(data = plot_data, aes(x = Date, y = Actual), color = "darkb
lue", size = 0.8, linetype = "solid") +
```

```

geom_line(data = plot_data, aes(x = Date, y = Fitted), color = "red",
size = 0.5, linetype = "solid") +
geom_line(data = forecasted_values_df, aes(x = Date, y = `Point Forec
ast`), color = "darkgreen", size = 0.5, linetype = "solid") +
geom_ribbon(data = forecasted_values_df, aes(x = Date, ymin = `Lo 95`,
ymax = `Hi 95`), alpha = 0.5, fill = "lightgrey") +
labs(title = "",
x = "Time",
y = "Water amount") +
theme_classic()

```

```

## Warning: Using `size` aesthetic for lines was deprecated in ggplot2
3.4.0.
##  Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warnin
g was
## generated.

```

ARIMA_p1

