Time Series Analysis with R

Imran Muhammad

November 2025

Contents

Practical relevance

- Time series analysis is a powerful statistical tool for examining data collected over time to identify **patterns**, **trends**, and **seasonal variations**.
- It enables researchers, policymakers, and businesses to understand dynamic changes and make accurate forecasts for **effective planning and decision-making**.
- In **public health**, time series analysis helps track and predict infectious disease outbreaks.
- In **meteorology**, it supports weather forecasting and disaster management.
- In **economics and business**, models like **ARIMA**, **SARIMA**, **Holt-Winter** and **State Space** are used for forecasting, risk assessment, and policy planning based on trends in inflation and employment.

R programming

- The **R programming language** is gaining significant attention among researchers and data scientists due to its flexibility and analytical power.
- R is supported by hundreds of packages that facilitate implementation across various disciplines such as statistics, economics, biology, finance, and engineering.
- In **statistical analysis**, R is widely used for data wrangling, visualization, and model development and analysis.
- Its open-source nature, strong community support, and integration with modern analytical tools make it an essential language for both academic research and applied data science.

The primary objective of today's presentation is to demonstrate the implementation of the `forecast` package in R for analyzing and forecasting time series data.

Special thanks to **Rob J. Hyndman** and his team for developing this outstanding R package [1].

An overview

- This presentation provides a comprehensive demonstration of time series analysis using `R`.
- Converting data into a time series object using the `ts()` function.
- An overview of the **ARIMA** model.
- Visualization using `tsdisplay()` and `autoplot()` functions. Implementing `auto.arima()`, `forecast()`, and `accuracy()` functions on COVID data.
- Fitting the **SARIMA** model to U.S. accidental death data with seasonality detection test.
- Includes **smoothing techniques** such as Holt, Brown, and Winters methods.
- Discusses **ETS** model.
- Explains statistical testing for **stationarity**, **linearity**, and **seasonality**.
- Demonstrates **model comparison** and **accuracy evaluation** using real-world data.

Time series and its frequency

- A **time series** is a sequence of observations recorded over regular time intervals (e.g., daily, monthly, yearly).
- The `frequency` defines how many observations occur in one unit of time (e.g., 12 for monthly, 4 for quarterly).
- In R, the `ts()` function creates time series objects by specifying data, start time, and frequency.
- Correctly defining `frequency` helps in accurate plotting, modeling, and forecasting of time series data. It is important in seasonal time series analysis.

Syntax:

```
ts(data, start = c(year, period), frequency = n)
```

Common frequencies

- Common frequencies for different data types are shown below:

Data Type	**Frequency**	**Explanation**
Yearly	1	One observation per year
Half-yearly	2	Two observations per year
Quarterly	4	Four observations per year
Monthly	12	Twelve observations per year
Weekly	52	52 weeks per year
Daily	365	365 days per year

Micro-level data

Micro-level time series data refer to observations collected at **very short time intervals** (such as seconds or minutes), capturing detailed and high-frequency economic behavior within a system.

Data Type	Frequency	Explanation
Yearly (minutes)	525,600	$60 \times 24 \times 365$ minutes per year
Daily (minutes)	1,440	60×24 minutes per day
Hourly	24	24 hours per day (if your unit is hour)

- **Stock Prices per Minute** e.g., price of a company's share recorded every minute.
- **Exchange Rates per Minute** e.g., USD to PKR value every minute.
- **Sales Transactions per Minute** e.g., number of sales in a store each minute.
- **Electricity Usage per Minute** e.g., power consumption in a factory every minute.

ts() function

The `ts()` function creates a **time series object** in R. In the case of monthly time series data, the frequency parameter should be set to `12`. The argument `start = c(2022, 1)` means the series starts in **January 2022**.

Creating a monthly time series (2022-2024)

```
data <- c(120,135,150,160,155,170,165,180,175,190,200,210,
          130,145,155,165,160,175,170,185,180,195,205,215,
          140,150,160,170,165,180,175,190,185,200,210,220)

ts_data <- ts(data, start = c(2022, 1), frequency = 12)
```

Output:

```
     Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
2022 120 135 150 160 155 170 165 180 175 190 200 210
2023 130 145 155 165 160 175 170 185 180 195 205 215
2024 140 150 160 170 165 180 175 190 185 200 210 220
```

Quarterly data

Creating a quarterly time series (2022-23)

```
data <- c(10, 12, 13, 15, 18, 20, 21, 24)
ts_data <- ts(data, start = c(2022, 1), frequency = 4)
```

Output

```
     Qtr1 Qtr2 Qtr3 Qtr4
2022   10   12   13   15
2023   18   20   21   24
```

R Code

```
ts_data <- ts(data, start = c(2022, 2), frequency = 4)
```

```
     Qtr1 Qtr2 Qtr3 Qtr4
2022         10   12   13
2023   15    18   20   21
2024   24
```

Yearly data

- For **yearly data**, set `frequency = 1`.
- The argument `start = c(2015)` means the series starts in the year **2015**.
- Useful for annual reports, production, or economic indicators.

Example:

Creating a yearly time series (2015-2024)

```
data <- c(120, 135, 150, 160, 155, 170, 175, 190, 200, 210)

ts_data <- ts(data, start = c(2015), frequency = 1)
ts_data
```

Output

```
Time Series:
Start = 2015
End = 2024
Frequency = 1
 [1] 120 135 150 160 155 170 175 190 200 210
```

ARIMA Model

- **ARIMA** is a popular model for analyzing and forecasting time series data. The model captures **both trends and short-term dependencies in data**. It is mainly used when the data become **stationary** after differencing.
- It combines three components:
  - **AR (p)** Autoregressive part: relationship between an observation and its previous values.
  - **I (d)** Integrated part: differencing to make the series stationary.
  - **MA (q)** Moving Average part: dependency on past forecast errors.
- The general form is denoted as `ARIMA(p, d, q)`.

$$\underbrace{c + \phi_1 y_{t-1} + \cdots + \phi_p y_{t-p}}_{\text{AR Component}} + \underbrace{\theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q}}_{\text{MA Component}} + \varepsilon_t$$

$$y_t = (\text{AR terms}) + (\text{MA terms}) + \text{error term}$$

Assumptions

- **Stationarity:** Mean and variance do not change over time.

$$E(y_t) = \mu, \quad Var(y_t) = \sigma^2$$

- **Linearity:** The model is a linear function of past values and errors.

$$y_t = c + \sum_{i=1}^{p} \phi_i y_{t-i} + \sum_{j=1}^{q} \theta_j \varepsilon_{t-j} + \varepsilon_t$$

- **Independence of Errors:** Residuals are uncorrelated.

$$Cov(\varepsilon_t, \varepsilon_{t-k}) = 0, \quad k \neq 0$$

- **Homoscedasticity:** Errors have constant variance over time.

$$Var(\varepsilon_t) = \sigma^2$$

COVID-19 Confirmed Cases

The datset represents 122 **daily** observations from April 16, 2021 to August 15, 2021. The data were recently used by [2]:

Data vector

```
x <- c(4976, 6127, 5152, 5445, 5499, 5857, 5870, 5908, 5611, 4825,
       4487, 5292, 5480, 5113, 4696, 4414, 4213, 3377, 4113, 4198,
       4298, 4109, 3785, 3447, 3084, 2869, 3265, 2517, 1531, 2379,
       3232, 2566, 3256, 4207, 2370, 4007, 3084, 3060, 2253, 2724,
       2726, 2482, 2455, 2697, 2117, 1771, 1843, 2028, 1893, 1923,
       1629, 1490, 1383, 1118, 1303, 1303, 1194, 1239, 1019, 838,
       1038, 1119, 1043, 991, 1050, 907, 663, 930, 1097, 1052, 935,
       901, 914, 735, 979, 1037, 1277, 1400, 1228, 1347, 830, 1517,
       1683, 1737, 1828, 1980, 1808, 1590, 1980, 2545, 2327, 2783,
       2607, 2452, 2145, 2579, 2158, 1425, 1841, 2819, 3752, 3262,
       4119, 4497, 4537, 4950, 5026, 4858, 3582, 4722, 5661, 4745,
       4720, 4455, 4040, 3884, 4850, 4934, 4619, 4786, 3711, 3669)
```

Creating daily time series

R code

```
x_ts <- ts(x, start = c(2021, 106), frequency = 365)
x_ts
```

Output

```
Time Series:
Start = c(2021, 106)
End   = c(2021, 227)
Frequency = 365
```

- The argument `frequency = 365` means the data are recorded **daily** within a year.
- The `start = c(2021, 106)` indicates April 16, 2021 (the 106th day of the year).
- The series ends on the 227th day August 15, 2021.

tsdisplay() function

The `tsdisplay() function` from the **forecast** package provides a quick and comprehensive visual summary of a time series. **It displays three diagnostic plots**. It is particularly useful for identifying trends, seasonality, and autocorrelation structures. This combined view helps determine suitable $ARIMA(p, d, q)$ model parameters.

R code

```
library(forecast)
tsdisplay(x_ts, main = "Daily Time Series with ACF and PACF")
```

1. **Time Series Plot:** shows trend and fluctuations over time.
2. **ACF:** shows correlation between current and lagged values.
3. **PACF:** shows direct correlation at specific lags after removing earlier effects.

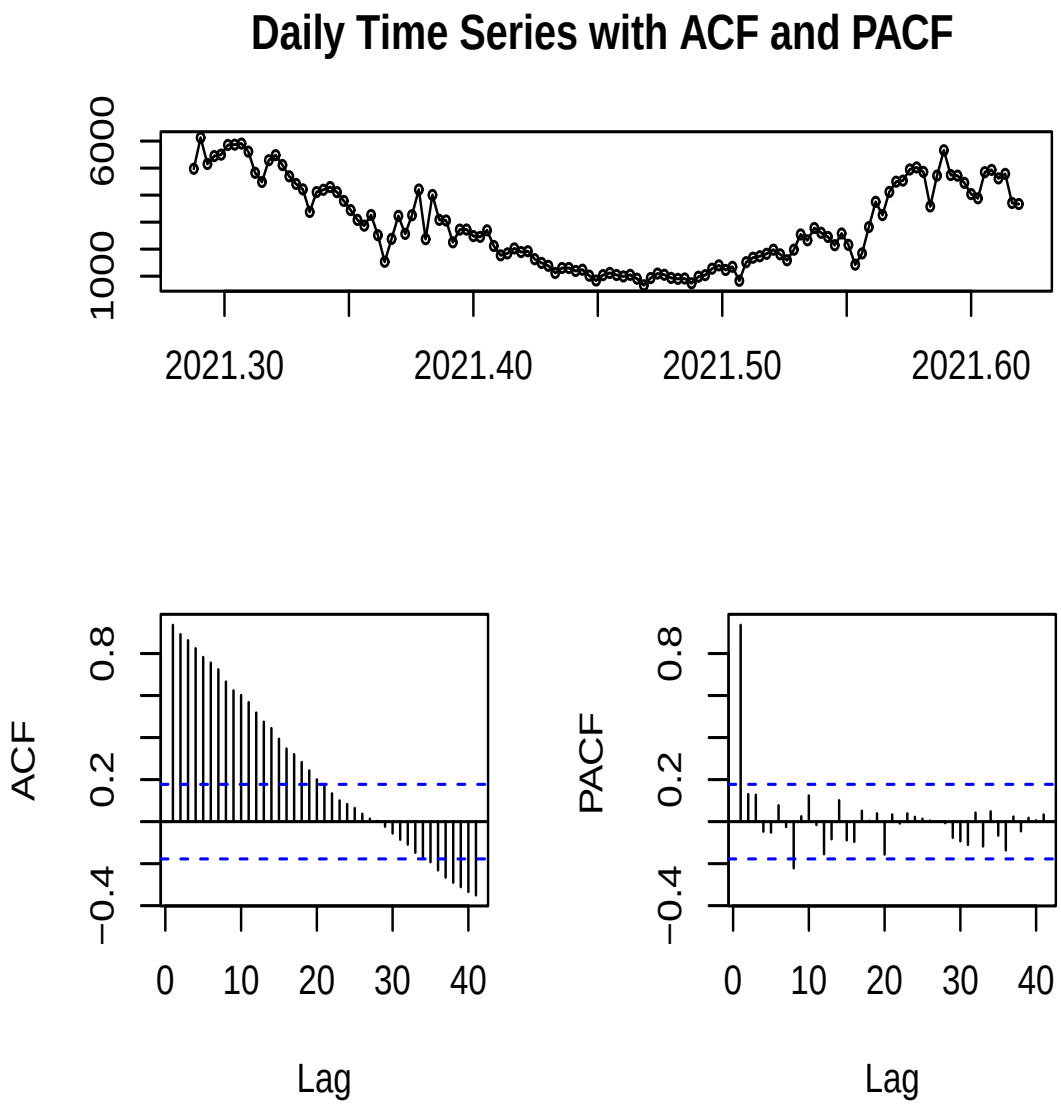

Figure : Time series, ACF, and PACF plots of daily confirmed COVID-19 cases in Pakistan.

diff() function

Concept

First differencing is used to remove a trend and make a time series stationary. If x_t is the original series, the first difference is:

$$z_t = x_t - x_{t-1}$$

```
z <- diff(x_ts)
tsdisplay(z)
```

- **The time series plot** of z_t (after first difference) shows reduced trend, indicating increased stationarity.
- The **ACF** shows reduced long-term correlation, suggesting successful differencing.
- The **PACF** helps identify potential autoregressive terms for ARIMA modeling.

After first differencing

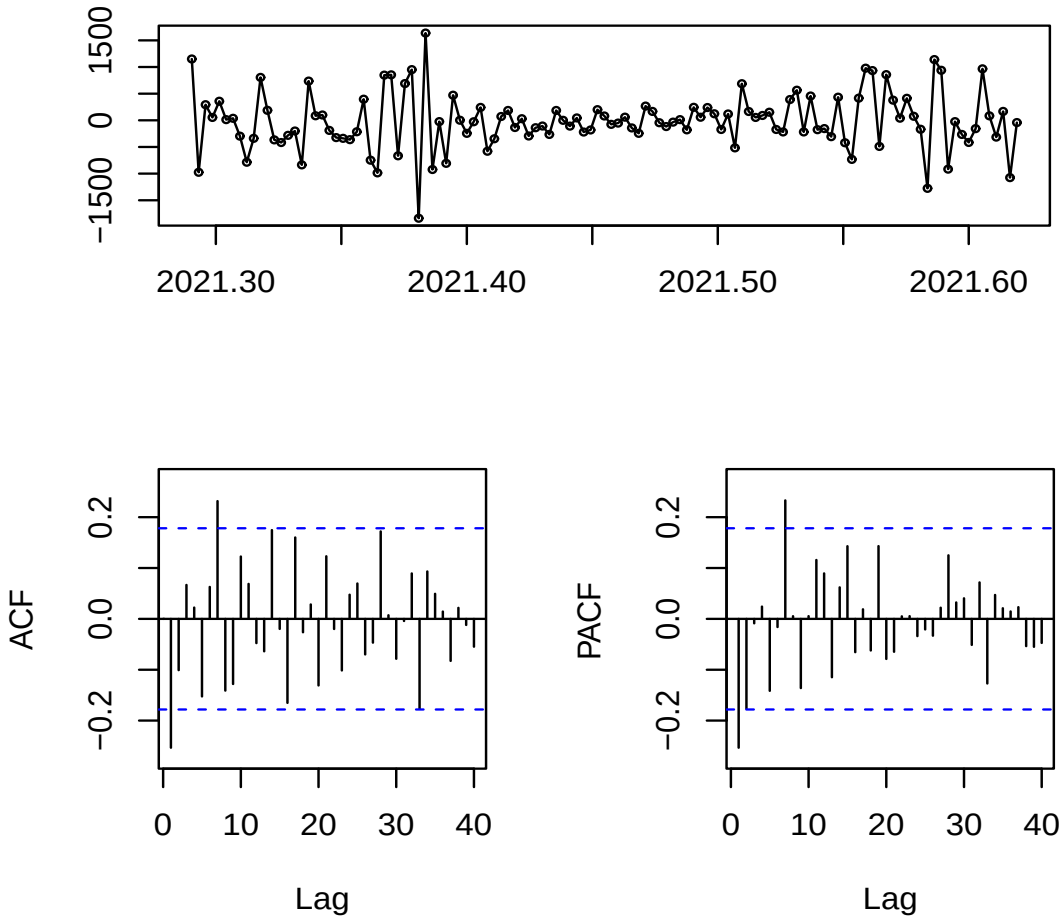

Figure : Plot of COVID-19 cases after first differencing.

adf.test() function

The Augmented DickeyFuller (ADF) test is used to check whether a time series is stationary. The null hypothesis (H_0) assumes non-stationarity, while the alternative (H_1) indicates stationarity.

R Code

```
install.packages("tseries")
library(tseries)

# Apply Augmented Dickey-Fuller test
adf.test(x_ts)
Dickey-Fuller = -1.4559, Lag order = 4, p-value = 0.8026
```

Interpretation

The ADF test checks whether the series is stationary. Since the p-value (0.8026) is greater than 0.05, we fail to reject the null hypothesis. Hence, the series is **non-stationary**.

Tsay.test() function

The Tsay linearity test is used to check whether a time series follows a linear or non-linear pattern. H_0: Series is linear H_1: Series is non-linear If p-value $<$ 0.05, reject H_0 (non-linear).

R Code

```
install.packages("TSA")
library(TSA)

# Apply Tsay test for non-linear
Tsay.test(x_ts, order = 1)
# Output
    Test Statistic: 0.9592
    p-value: 0.3294
```

Since $p > 0.05$, we **fail to reject** H_0.
Conclusion: There is no evidence of non-linearity.

auto.arima() function

To automatically fit the ARIMA model, we use the `auto.arima()` function as shown below:

R Code

```
fit <- auto.arima(x_ts)
fit
```

Output

ARIMA(2,1,0)

		Estimate	Std. Error
Coefficients:	ar1	-0.3035	0.0905
	ar2	-0.1841	0.0930

σ^2 = 253631 Log Likelihood = -923.59 AIC = 1853.18, AICc = 1853.39, BIC = 1861.57

fitted() function

```
# Plot actual time series
plot(x_ts, xlab = "Time (daily)", ylab = "Confirmed cases",
     col = 1, lwd = 2, lty = 1)

# Add ARIMA fitted values
lines(fitted(fit), col = "red", lwd = 2, lty = 2)
```

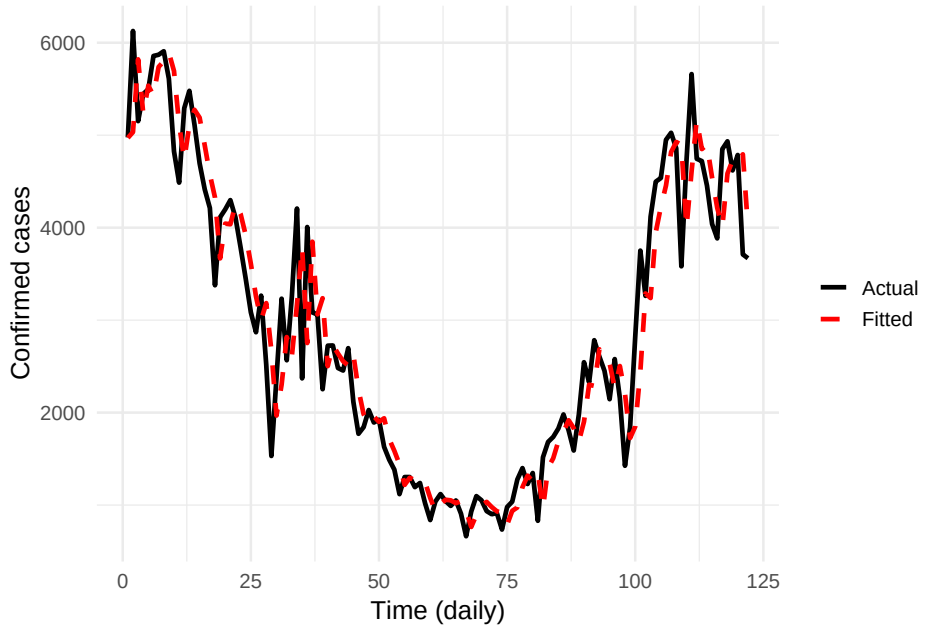

Figure : Actual and fitted plot.

forecast() function

The `forecast()` function provides forecasts for the next h periods. It is as follows:

R Code

```
# Forecast 5 steps ahead
fc <- forecast(fit, h = 5)
fc
```

Forecast Table

Time	Point Forecast	Lo 80	Hi 80	Lo 95 / Hi 95
2021.6219	3879.70	3234.29	4525.11	2892.63 / 4866.78
2021.6247	3823.49	3036.95	4610.03	2620.59 / 5026.39
2021.6274	3801.75	2923.77	4679.73	2459.00 / 5144.50
2021.6301	3818.70	2834.74	4802.66	2313.86 / 5323.54
2021.6329	3817.56	2740.21	4894.91	2169.89 / 5465.23

autoplot() function

```
autoplot(fc)
```

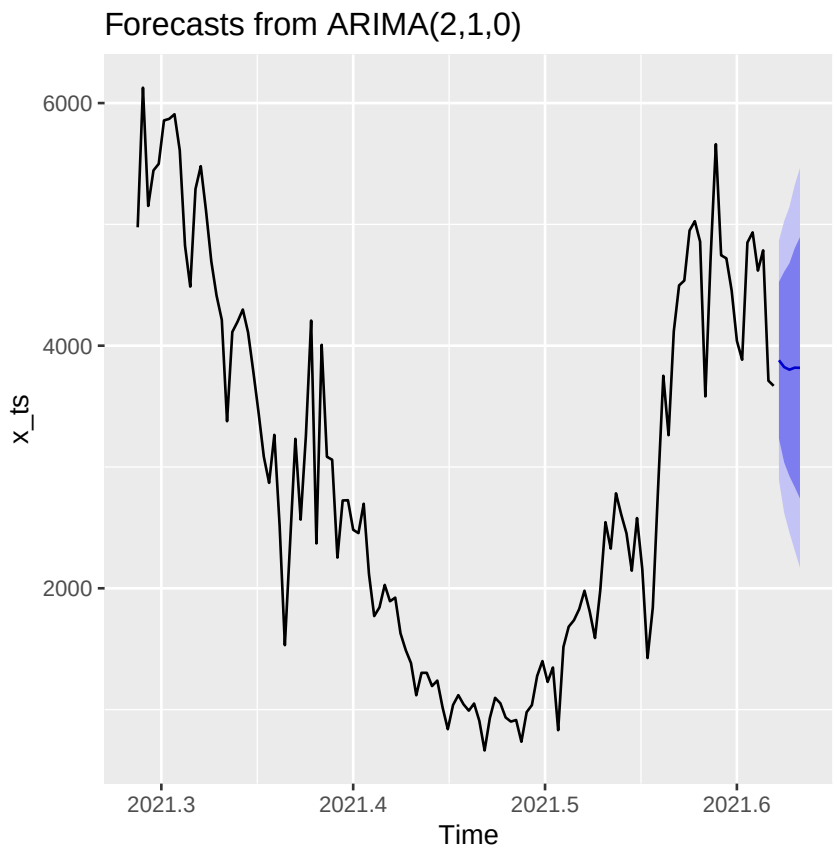

Figure : Actual and a five days ahead forecast plot.

accuracy() function

Another useful function is `accuracy()`, which provides several accuracy measures. The most commonly used ones include Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and Mean Absolute Percentage Error (MAPE).

R Code

```
accuracy(fit)
```

- **ME (Mean Error):** -14.93
- **RMSE (Root Mean Squared Error):** 497.39
- **MAE (Mean Absolute Error):** 368.76
- **MPE (Mean Percentage Error):** -2.62%
- **MAPE (Mean Absolute Percentage Error):** 14.35%

ggtsdisplay() function

- Now, we use a seasonal data to fit a **SARIMA** model.
- The `ggtsdisplay()` function is used to visually inspect the series, similar to `tsdisplay()`.
- Before using `ggtsdisplay()`, ensure that the `ggplot2` package is installed.
- The **USAccDeaths** dataset contains monthly accidental deaths in the USA from 1973 to 1978 and is available in the `forecast` package.

R Code

```
library(ggplot2)

# Display the time series, ACF, and PACF
ggtsdisplay(USAccDeaths)
```

USAccDeaths display

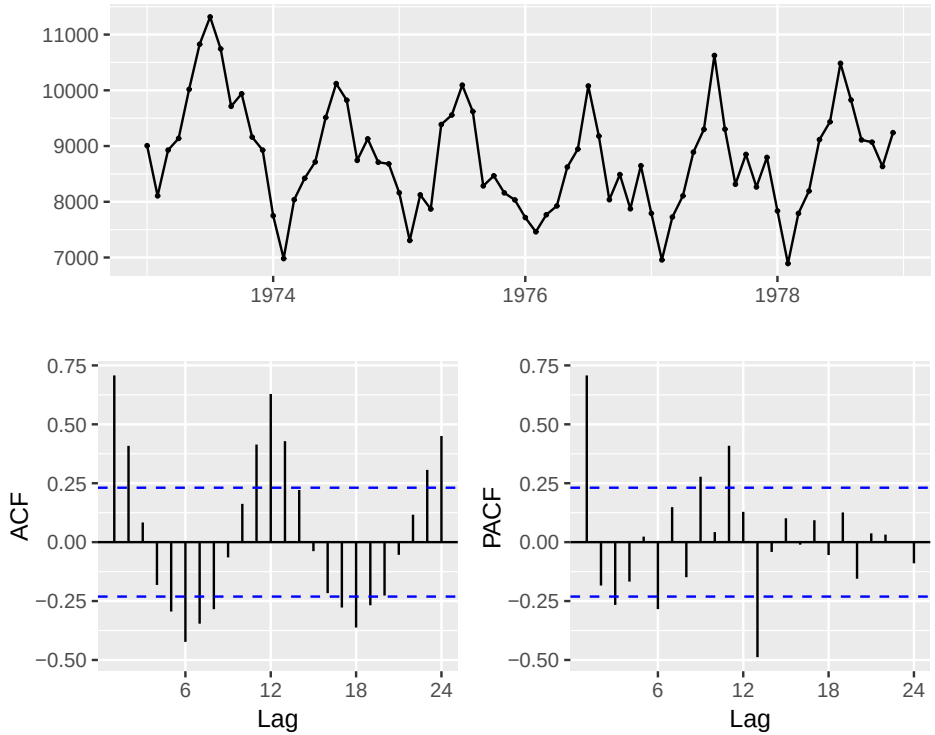

Figure : Time series, ACF, and PACF plots.

seasonal = TRUE

The argument `seasonal = TRUE` in the `auto.arima()` function enables to model seasonal data. The Seasonal ARIMA (SARIMA) model is denoted as SARIMA$(p, d, q)(P, D, Q)[s]$, where p, d, q and P, D, Q are the non-seasonal and seasonal parameters, respectively, corresponding to the seasonal period s.

R Code

```
fit <- auto.arima(USAccDeaths, seasonal = TRUE)
fit
```

Output

Series: USAccDeaths SARIMA(0,1,1)(0,1,1)[12]

		Estimate	Std. Error
Coefficients:	ma1	-0.4303	0.1228
	sma1	-0.5528	0.1784

Log Likelihood = -425.44 AIC = 856.88, AICc = 857.32, BIC = 863.11

SARIMA(0,1,1)(0,1,1)[12]

Model Components

Component	Value	Meaning
p	0	Non-seasonal AR term (none)
d	1	Non-seasonal differencing to remove trend
q	1	Non-seasonal MA term
P	0	Seasonal AR term (none)
D	1	Seasonal differencing to remove yearly seasonality
Q	1	Seasonal MA term
s	12	Seasonal period (12 months)

`Interpretation:` Trend is removed using first differencing ($d = 1$), seasonality is captured through seasonal differencing ($D = 1$) and a seasonal MA term ($Q = 1$). The model includes only moving average components, with no autoregressive terms.

Actual and fitted plots

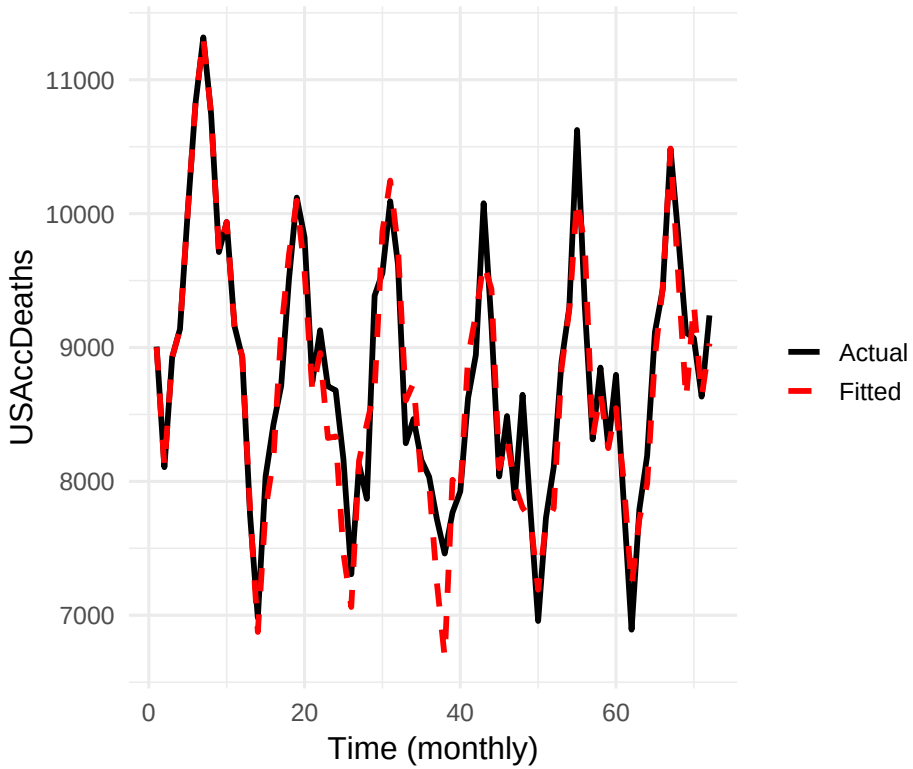

Figure : Actual and fitted plots.

Actual and forecast plots

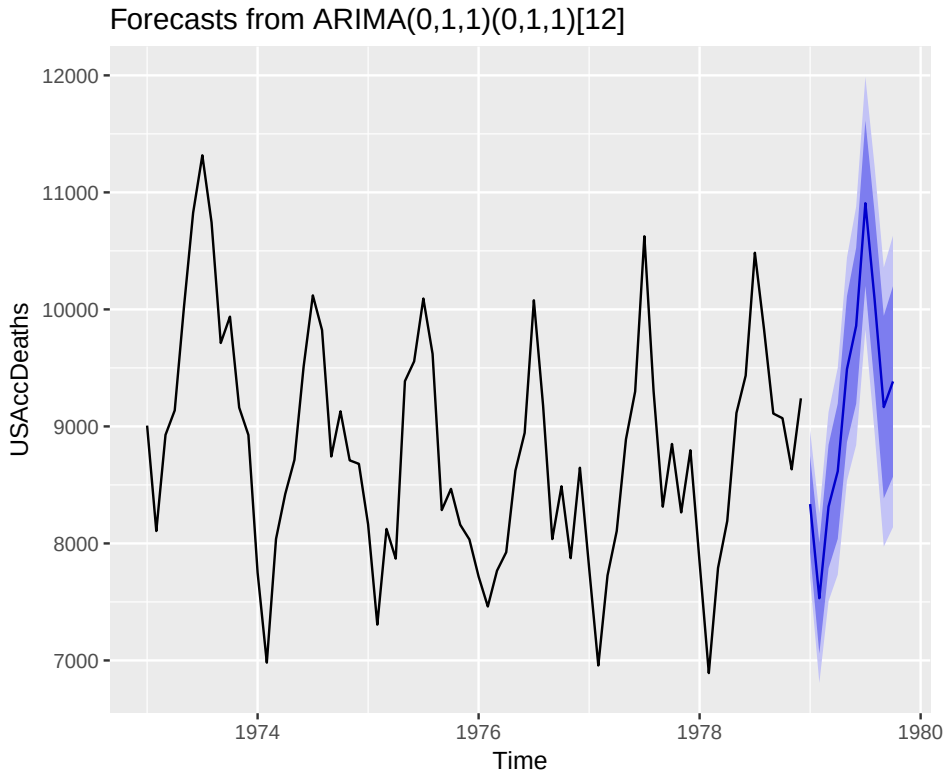

Figure : Actual and 10-month ahead forecasts.

A graphic way to check Seasonality

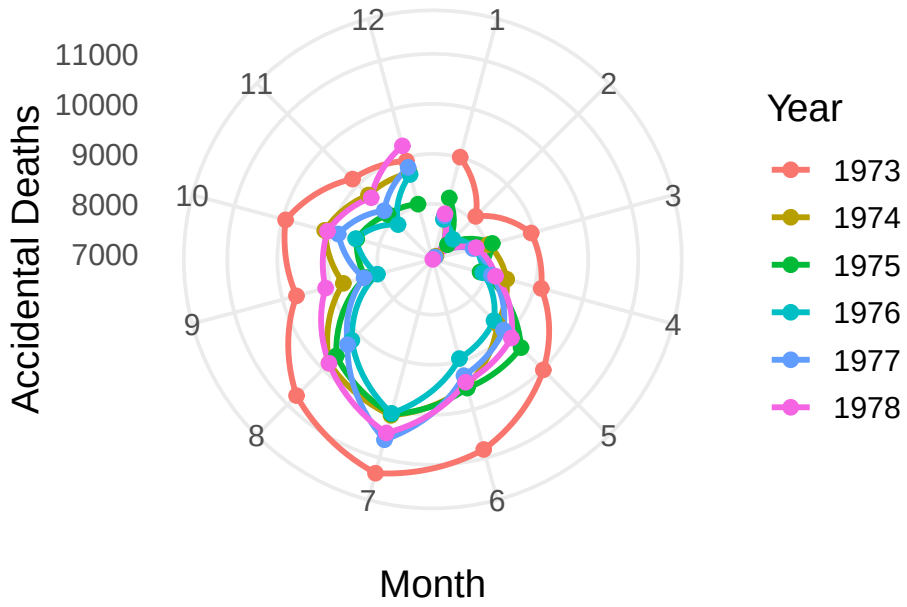

Figure : A polar plot.

isSeasonal() function

R Code

```
library(seastests)

# Test for seasonality
isSeasonal(USAccDeaths, test = "combined",
freq = frequency(USAccDeaths))
```

Result

```
TRUE
```

Interpretation: The USAccDeaths series exhibits significant seasonality.

ETS Models Overview

The ETS time series model stands for Error, Trend, and Seasonality it is a family of exponential smoothing state-space models used for forecasting.

- ETS models combine **level, trend, seasonality, and error** in flexible ways.
- They are widely used for **forecasting time series** with or without trend/seasonality.
- An ETS model can be written as follows:
  - ETS(A,N,N) additive errors, no trend, no seasonality
  - ETS(M,N,N) multiplicative errors, no trend, no seasonality
  - ETS(A,A,A) additive errors, additive trend, additive seasonality

There are 30 $(2 \times 5 \times 3)$ theoretical ETS model's combinations, but only 15 are used in practice. For further mathematical details and derivations, see [4, 5].

Combinations

Table : Classification of Exponential Smoothing Methods

Trend Component	**Seasonal Component**		
	N	A	M
N	N, N	N, A	N, M
A	A, N	A, A	A, M
A_d	A_d,N	A_d,A	A_d,M
M	M, N	M, A	M, M
M_d	M_d,N	M_d,A	M_d,M

Note: The notation (e.g., A,A) represents combinations of additive trend and additive seasonal components.

ETS(A,N,N)

Model: ETS(A,N,N) Simple Exponential Smoothing

Components: Additive error, No trend, No seasonality

Model Equations

$$\text{Forecast: } \hat{y}_{t+h|t} = \ell_t$$

$$\text{Level update: } \ell_t = \ell_{t-1} + \alpha(y_t - \ell_{t-1})$$

- No trend, no seasonality.
- One-step-ahead forecast equals the current level: $\hat{y}_{t+1|t} = \ell_t$.
- α is the smoothing parameter, $0 < \alpha < 1$.

ETS(A,A,N)

Model: ETS(A,A,N) Holts Linear Trend

Components: Additive error, Additive trend, No seasonality

Model Equations

$$\text{Forecast: } \hat{y}_{t+h|t} = \ell_t + hb_t$$

$$\text{Level: } \ell_t = \ell_{t-1} + b_{t-1} + \alpha e_t$$

$$\text{Trend: } b_t = b_{t-1} + \beta e_t$$

$$\text{where } e_t = y_t - (\ell_{t-1} + b_{t-1})$$

- Captures both **level** and **trend** components.
- Suitable for data with a linear trend.
- α and β are smoothing parameters, $0 < \alpha, \beta < 1$.
- When $\beta = 0$, the model reduces to Simple Exponential Smoothing.

ETS(A,Ad,N)

Model: ETS(A,Ad,N) Damped Trend Model

Components: Additive error, Additive damped trend, No seasonality

Model Equations

$$\text{Forecast: } \hat{y}_{t+h|t} = \ell_t + b_t \frac{1-\phi^h}{1-\phi}$$

$$\text{Level: } \ell_t = \ell_{t-1} + \phi b_{t-1} + \alpha e_t$$

$$\text{Trend: } b_t = \phi b_{t-1} + \beta e_t$$

$$\text{where } e_t = y_t - (\ell_{t-1} + \phi b_{t-1})$$

- The damping parameter ϕ $(0 < \phi < 1)$ reduces the long-term trend effect.
- When $\phi = 1$, the model becomes Holts Linear Trend (ETS(A,A,N)).
- Suitable for time series with trends that gradually level off.
- α and β are smoothing parameters.

ets() function

Here, we use the COVID-19 time series data

R Code

```
fit <- ets(x_ts)
fit
```

Output

ETS(M,N,N)
Smoothing parameters: alpha = 0.6255
Initial state: l = 5171.1248
Sigma: 0.1862
Model Selection Criteria: AIC = 2088.543, AICc = 2088.747, BIC = 2096.955

Description

- **Model Type:** ETS(M,N,N)
  - M Multiplicative error (errors scale with level)
  - N No trend
  - N No seasonality
- **Smoothing Parameter:** $\alpha = 0.6255$
  - Determines how quickly the level updates with new data
  - 62.55% weight is given to the latest observation
- **Initial Level:** $\ell_0 = 5171.1248$
  - Baseline starting value of the series
- **Error Standard Deviation:** $\sigma = 0.1862$
  - Measures magnitude of multiplicative errors
  - Indicates variability around fitted values

Comparison

ARIMA(2,1,0)

- **Model Selection:** AIC = 1853.18, AICc = 1853.39, BIC = 1861.57

ETS(M,N,N)

- **Model Selection:** AIC = 2088.543, AICc = 2088.747, BIC = 2096.955

The following Table presents the accuracy measures of the fitted models:

Table : Forecast accuracy comparison between ARIMA and ETS models.

Model	**RMSE**	**MAE**	**MAPE (%)**
ARIMA(2,1,0)	497.39	368.76	14.35
ETS(M,N,N)	500.13	374.03	14.55

Graphical representation

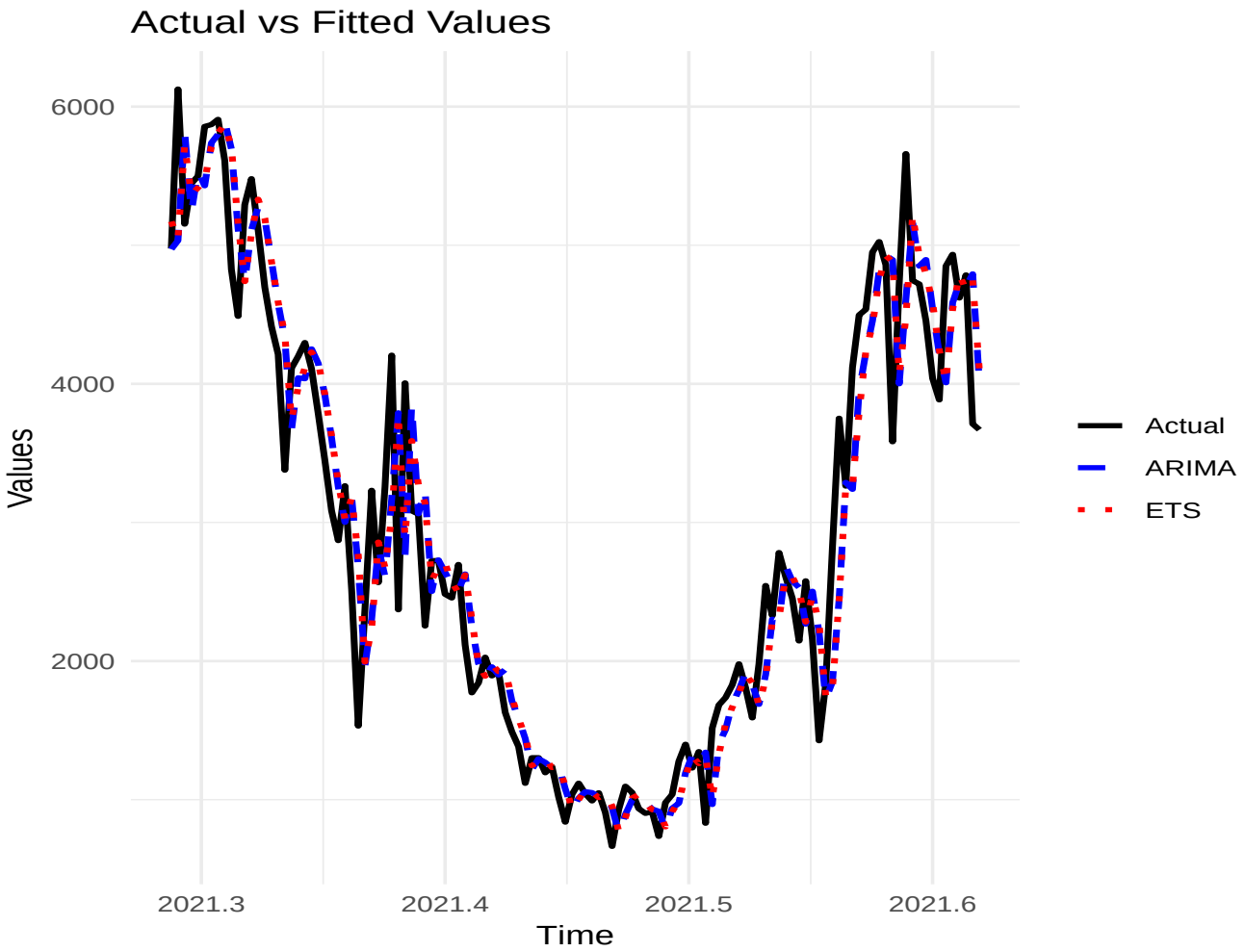

Figure : Graphical representation of actual versus predicted values.

Smoothing models

1. Brown or SES model

- Models data with a single component **level only**.
- Suitable for series with no trend and no seasonality.
- `brown <- HoltWinters(data, beta = FALSE, gamma = FALSE)`

2. Holts or DES model

- Extends Browns model by adding a **trend component**.
- Useful for data with linear upward or downward trend.
- `holt <- HoltWinters(data, gamma = FALSE)`

3. Holt-Winters or TES Model

- Incorporates both **trend and seasonality**.
- Can handle additive or multiplicative seasonal effects.
- `hw <- HoltWinters(data)`

Application

Here, we fit a Holt-Winters exponential smoothing model to the `USAccDeaths` dataset. Since the data exhibit seasonality, the models performance can be compared with a SARIMA model.

R Code

```
hw <- HoltWinters(USAccDeaths)
```

Output

Model: Holt-Winters exponential smoothing with trend and additive seasonality
Smoothing Parameters:
$\alpha = 0.7377, \ \beta = 0.0223, \ \gamma = 1.0000$
Coefficients:
Level $(a) = 8798.56$, Trend $(b) = -22.68$
Seasonal Components:
$s_1 = -802.45, \ s_2 = -1739.84, \ s_3 = -960.33,$
$s_4 = -594.42, \ s_5 = 259.30, \ s_6 = 673.55,$
$s_7 = 1770.77, \ s_8 = 1030.97, \ s_9 = 210.89,$
$s_{10} = 548.55, \ s_{11} = 127.69, \ s_{12} = 441.44$

Forecast

A 10-step ahead forecast using Holt-Winters model is as follows:

R code

```
fc <- forecast(hw, h = 10)
fc
```

Output

Month	Point	Lo 80	Hi 80	Lo 95	Hi 95
Jan 1979	7973.43	7489.49	8457.37	7233.31	8713.55
Feb 1979	7013.36	6407.22	7619.50	6086.35	7940.38
Mar 1979	7770.19	7058.51	8481.88	6681.76	8858.62
Apr 1979	8113.43	7306.22	8920.65	6878.91	9347.96
May 1979	8944.47	8048.47	9840.47	7574.15	10314.79
Jun 1979	9336.04	8356.09	10316.00	7837.33	10834.76
Jul 1979	10410.58	9350.30	11470.86	8789.03	12032.14
Aug 1979	9648.11	8510.31	10785.91	7908.00	11388.22
Sep 1979	8805.34	7592.24	10018.45	6950.06	10660.63
Oct 1979	9120.33	7833.70	10406.96	7152.59	11088.06

autoplot() function

```
autoplot(fc)
```

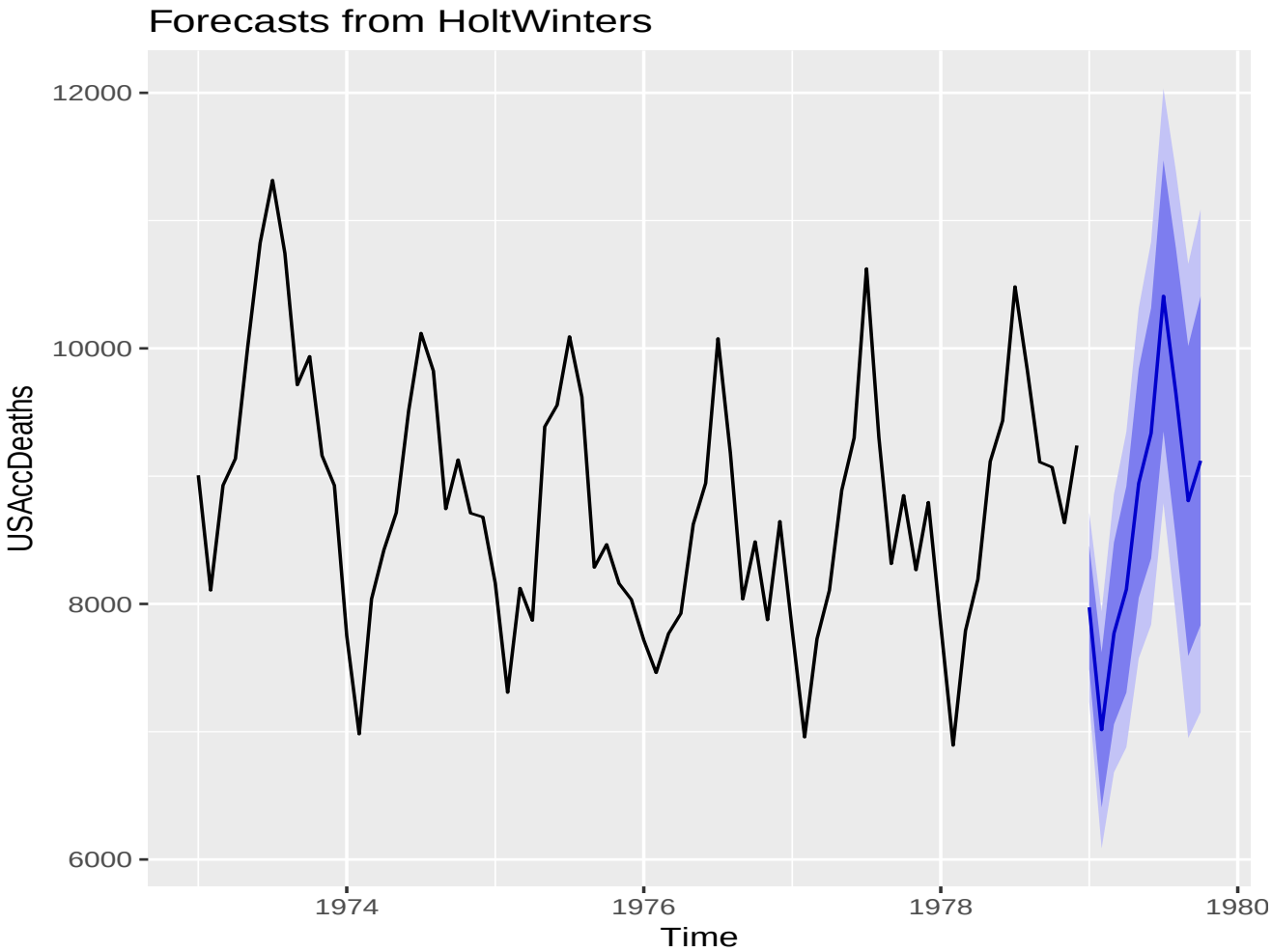

Figure : Actual and forecast plot.

Comparison of Holt-Winters and SARIMA Forecasts

R Code

```
# --- Load required libraries ---
library(forecast)
library(ggplot2)
library(dplyr)
library(zoo)   # for as.yearmon()

# --- Data ---
data("USAccDeaths")

# --- Holt-Winters model ---
hw <- HoltWinters(USAccDeaths)
fc_hw <- forecast(hw, h = 12)

# --- SARIMA model ---
sarima <- auto.arima(USAccDeaths)
fc_sarima <- forecast(sarima, h = 12)
```

Continued...

R Code

```
# --- Convert time series to data frames ---
ts_to_df <- function(ts_data, label) {
  data.frame(
    Date = as.Date(as.yearmon(time(ts_data))),  # convert time to Date
    Value = as.numeric(ts_data),
    Type = label
  )
}

# --- Prepare data frames for actual and forecast values ---
df_actual  <- ts_to_df(USAccDeaths, "Actual")
df_hw      <- ts_to_df(fc_hw$mean, "Holt-Winters")
df_sarima  <- ts_to_df(fc_sarima$mean, "SARIMA")

# --- Combine all data ---
df_all <- rbind(df_actual, df_hw, df_sarima)
```

Continued...

R Code

```
# --- Plot comparison using ggplot2 ---
ggplot(df_all, aes(x = Date, y = Value,
                   color = Type, linetype = Type)) +
  geom_line(size = 1.1) +
  labs(title = "Comparison of Holt-Winters and SARIMA Forecasts",
       x = "Year", y = "US Accident Deaths",
       color = "Model", linetype = "Model") +
  scale_color_manual(values = c("Actual" = "black",
                                "Holt-Winters" = "blue",
                                "SARIMA" = "red")) +
  theme_minimal(base_size = 14) +
  theme(plot.title = element_text(face = "bold", hjust = 0.5))
```

Visualization

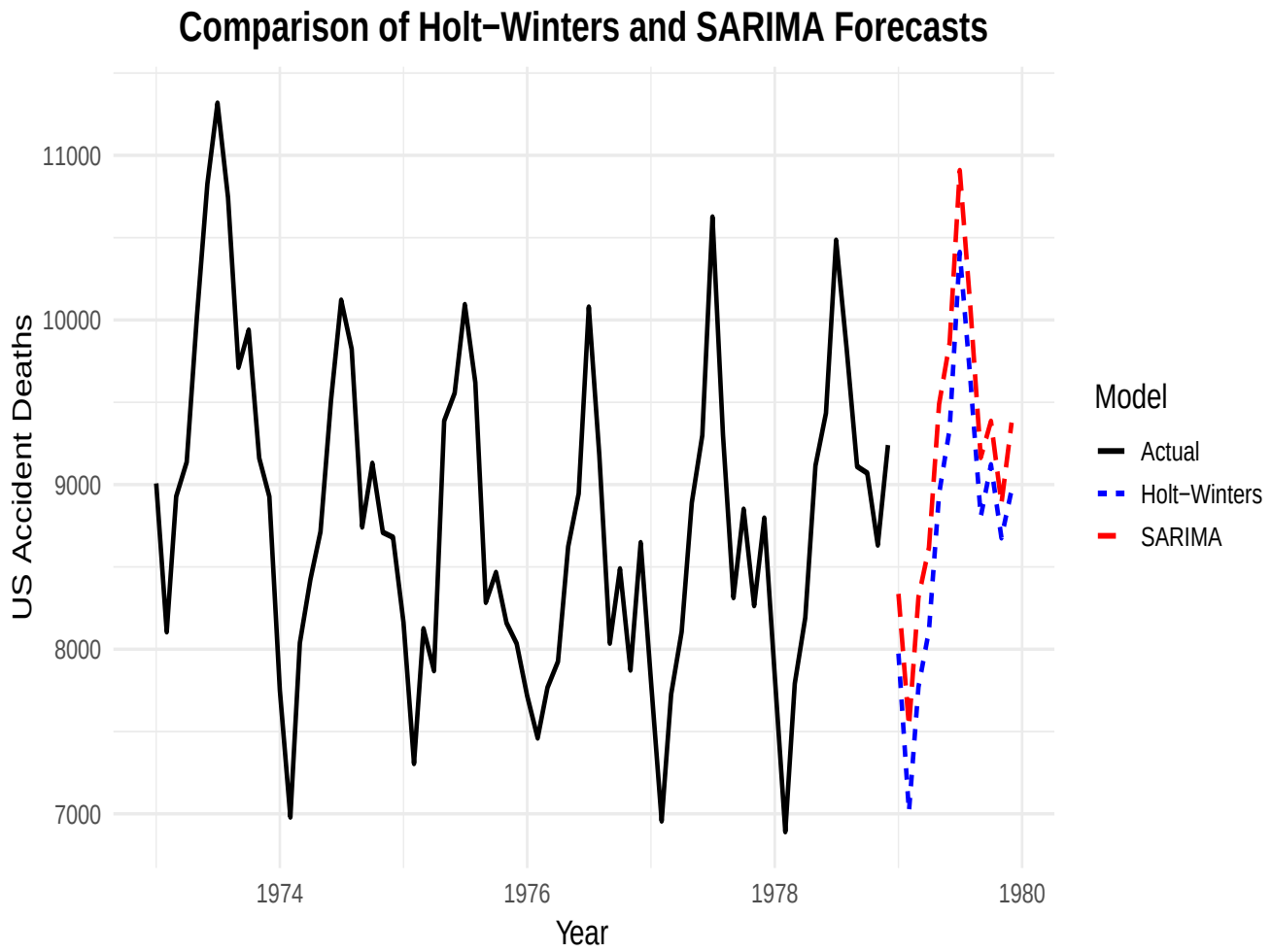

Figure : A polar plot of the series.

Comparison

Output

Holt-Winters Model:

$$\text{ME} = 61.39, \quad \text{RMSE} = 379.46, \quad \text{MAE} = 273.62, \quad \text{MPE} = 0.73, \quad \text{MAPE} = 3.21, \quad \text{M}$$

SARIMA Model:

$$\text{ME} = 58.84, \quad \text{RMSE} = 285.36, \quad \text{MAE} = 200.95, \quad \text{MPE} = 0.67, \quad \text{MAPE} = 2.35, \quad \text{M}$$

The **SARIMA model** performs better overall, as indicated by its lower RMSE, MAE, and MAPE values compared to the Holt-Winters model, suggesting more accurate forecasts.

checkresiduals() function

The diagnostic plots check whether residuals are:

- Residuals are uncorrelated (no autocorrelation)
- Residuals are approximately normally distributed
- Residuals have zero mean and constant variance (homoscedasticity)

If the residuals are uncorrelated, normally distributed, and have constant variance, they are considered **white noise**, implying a good model fit.

R Code for Residual Diagnostics

```
# Holt-Winters model diagnostics
checkresiduals(fc_hw)

#  SARIMA model diagnostics
checkresiduals(fc_sarima)
```

Holt-Winter diagnostics plot

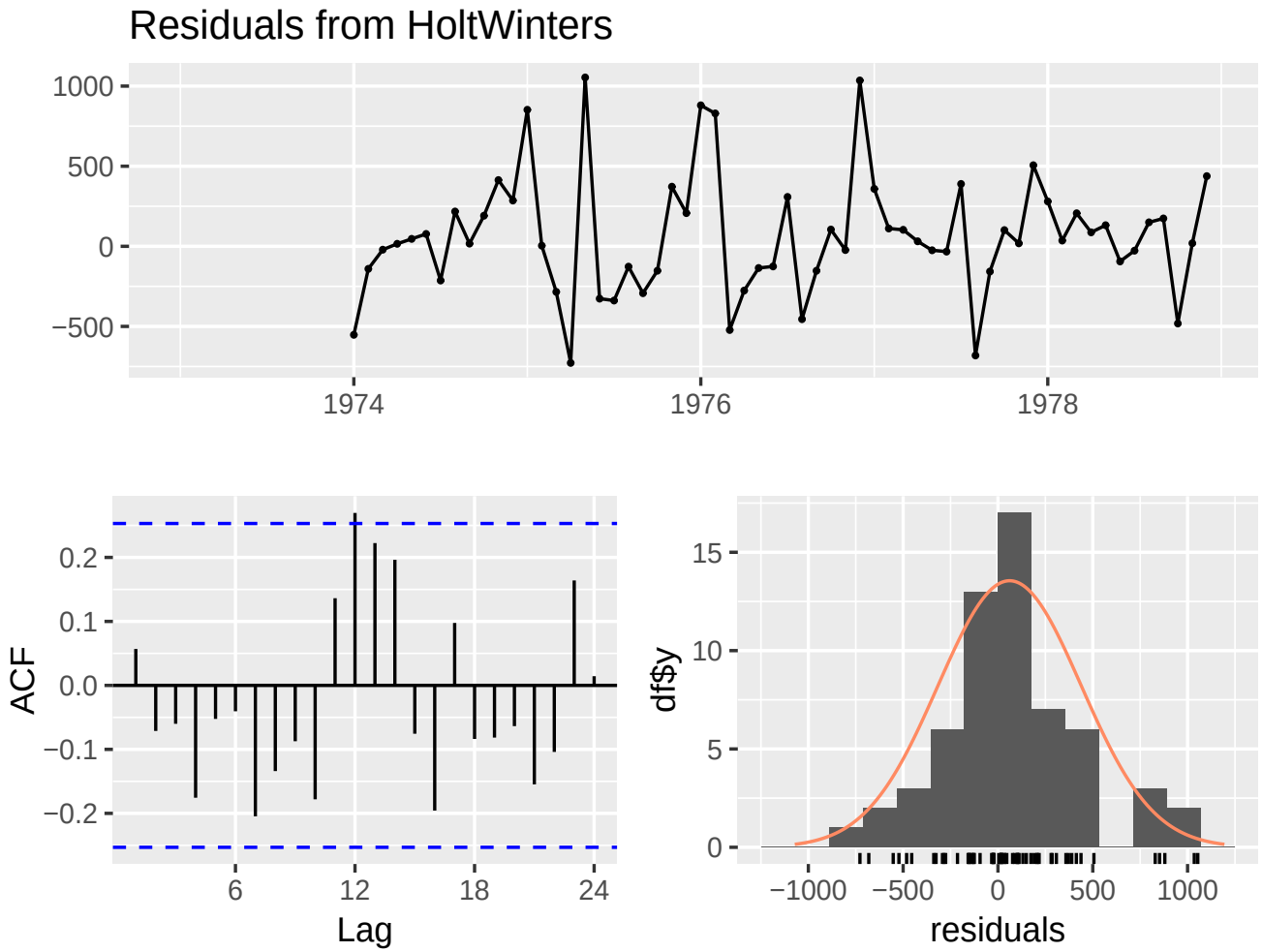

Figure : Diagnostic plot of the Holt-Winter model.

SARIMA diagnostics plot

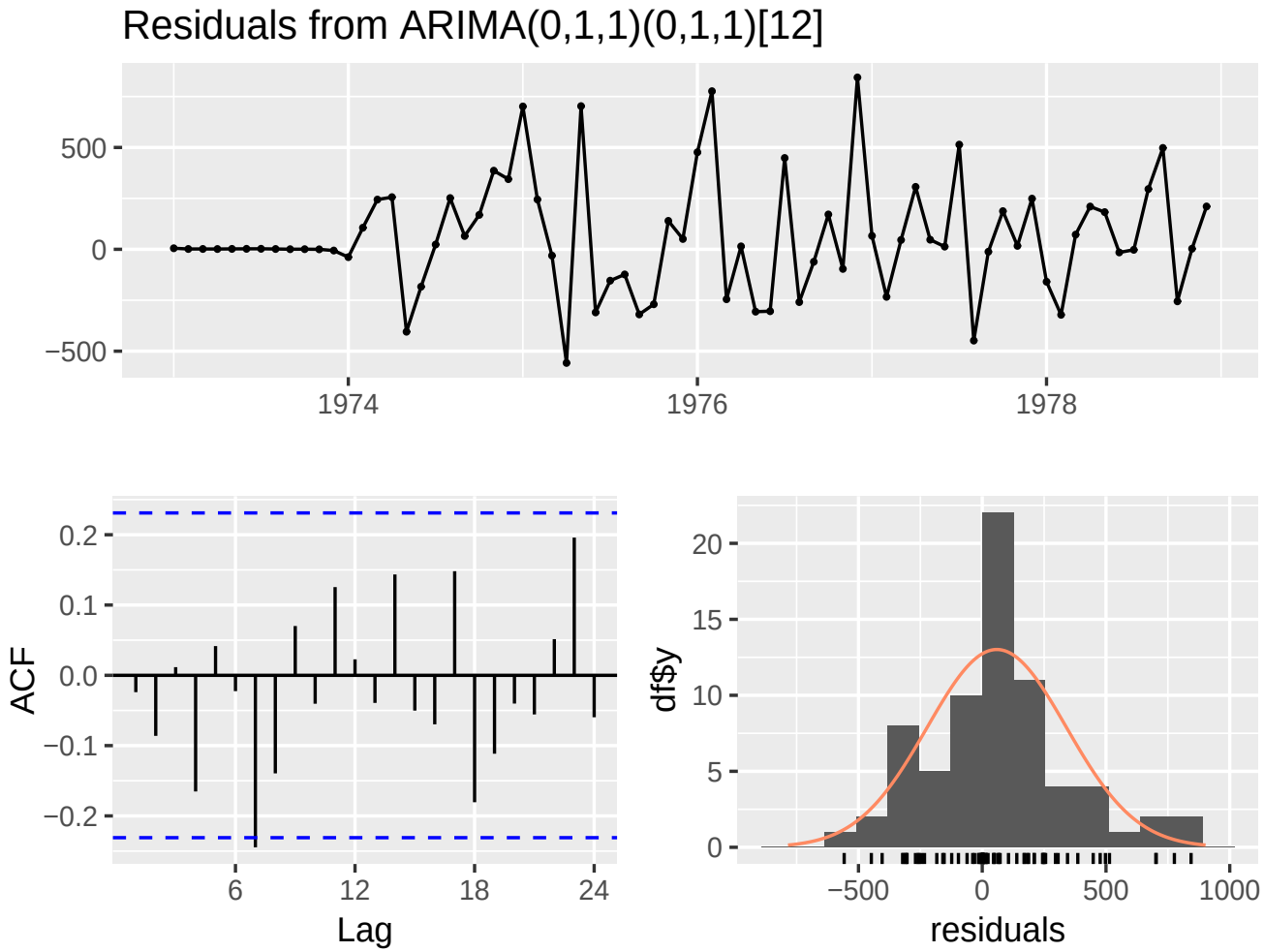

Figure : Diagnostic plot of the SARIMA model.

Ljung-Box test

The Ljung-Box test is used to check whether the residuals from a time series model are independently distributed (i.e., show no autocorrelation).

- H_0: The residuals are independently distributed (no autocorrelation).
- H_1: The residuals are not independently distributed (there is autocorrelation).

Output

- Model: ARIMA(0,1,1)(0,1,1)[12]
- $Q^* = 13.467$
- p-value = 0.336

Since the p-value (0.336) is greater than 0.05, we fail to reject the null hypothesis. Thus, there is no significant autocorrelation, indicating that the ARIMA(0,1,1)(0,1,1)[12] model fits the data adequately.

dm.test() function

The Diebold-Mariano (DM) test is used to statistically compare the predictive accuracy of two competing forecasting models.

- H_0: Both models have equal predictive accuracy.
- H_1: The predictive accuracy of the two models is significantly different.

R Code

```
# --- Load libraries ---
library(forecast)
library(Metrics)

# --- Compare forecast accuracy ---
dm_result <- dm.test(residuals(fc_hw),
                     residuals(fc_sarima),
                     alternative = "two.sided")

dm_result
```

Forecast accuracy comparison

DM Test Results (HoltWinters vs. SARIMA)

Output

- DM statistic = 2.6317
- p-value = 0.01041
- Alternative hypothesis: Two-sided

Interpretation

The p-value (0.01041) is less than 0.05, so we **reject the null hypothesis**. This means there is a **significant difference in forecast accuracy** between the HoltWinters and SARIMA models.
Since the SARIMA model showed lower forecast errors (RMSE and MAPE), it provides **more accurate forecasts** for the `USAccDeaths` data.

References

Hyndman, R. J., & Khandakar, Y. (2008). Automatic time series forecasting: The forecast package for R. Journal of Statistical Software, 27(3), 1-22.

Fayomi, A., Jamal Abdul Nasir, Algarni, A., Rasool, M. S., & Chesneau, C. (2022). *Best selected forecasting models for COVID-19 pandemic. Open Physics, 20*(1), 261–273.

Hyndman, R. J., & Athanasopoulos, G. (2021). *Forecasting: Principles and Practice* (3rd ed.).

Hyndman, R. J., Koehler, A. B., Snyder, R. D., & Grose, S. (2002). *A state space framework for automatic forecasting using exponential smoothing methods. International Journal of Forecasting, 18*(3), 439–454.

Hyndman, R. J., Akram, M., & Archibald, B. (2008). *The admissible parameter space for exponential smoothing models.* Annals of the Institute of Statistical Mathematics, 60(2), 407–426.

Thank You!

Questions and Discussion

imranshakoor84@yahoo.com