Samson, Justine Aizel D.
2021053281

Time Series Analysis
FA 1

United Kingdom Airline Miles Flown

1. Find the sample autocorrelation function for this time series.
    1. Is the seasonality apparent in the sample autocorrelation function?
    2. Is the time series stationary or nonstationary?

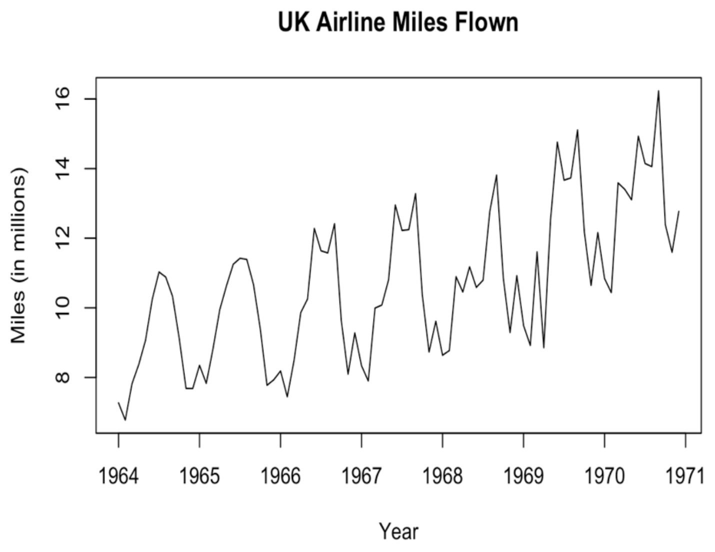
For all the questions, review the entire code at the GitHub link.

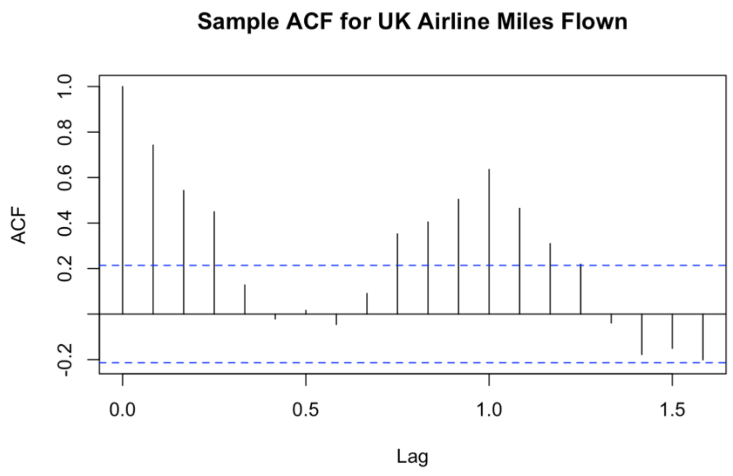
Explanation of some code:

```r
# Convert the data to a time series object
miles_ts <- ts(df$`Miles(in millions)`, start = c(1964, 1), frequency = 12)

# Plot the time series
plot(miles_ts, main = "UK Airline Miles Flown", ylab = "Miles (in millions)", xlab = "Year")

# Step 2: Find the sample autocorrelation function (ACF)
acf(miles_ts, main = "Sample ACF for UK Airline Miles Flown")
```

The `ts()` function in R is used to transform the "Miles (in millions)" column from a dataset into a series of values object. The data is set to begin in January 1964 and have a monthly frequency of 12 observations annually. Following the creation of the time series object {miles_ts}, it is displayed with a labeled y- and x-axis to visualize the trend of UK airline miles flown over time using the `plot ()` function. To help detect any patterns, trends, or seasonality in the time series, the {acf()} function lastly computes and plots the sample autocorrelation function (ACF), which shows the correlation of the time series data with its own lagged values.

Plot



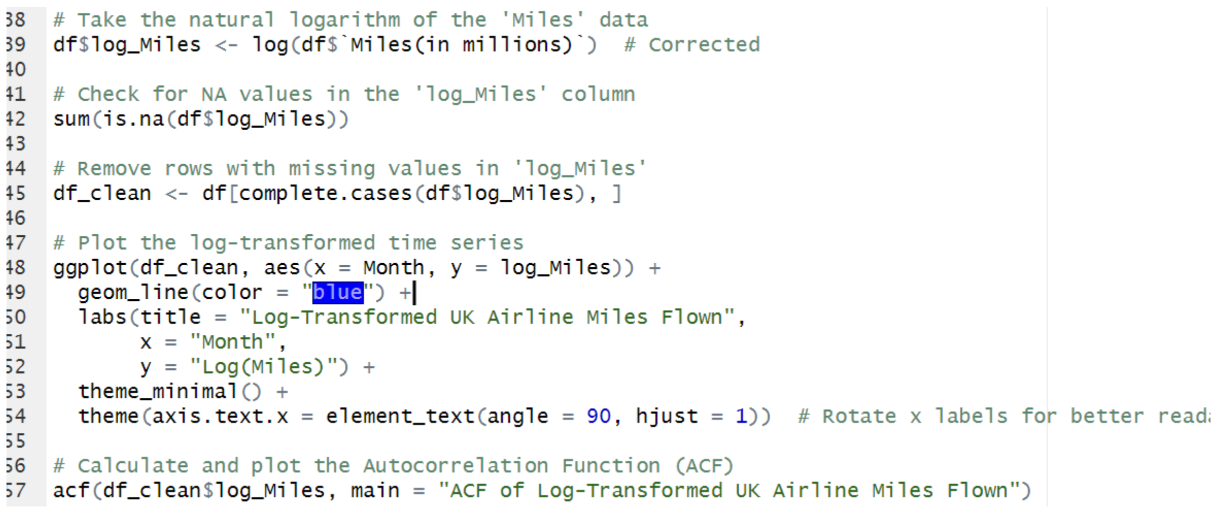


a. Seasonality is evident, indeed. Significant peaks can be seen in the ACF plot regularly, or around every 12 months. Given that the correlation coefficients rise and decrease periodically, which is characteristic of data influenced by seasonality, this suggests that the data follow a seasonal pattern.

b. Not stationary. The ACF is robust across long lags and does not fade quickly. The fact that the autocorrelation in this time series remains high for several lags, indicates that the series is nonstationary, in contrast to a stationary time series where the ACF would immediately decay towards zero after a few lags.

2. Take the natural logarithm of the data and plot this new time series.
    1. What impact has the log transformation had on the time series?
    2. Find the autocorrelation function for this time series.
    3. Interpret the sample autocorrelation function.

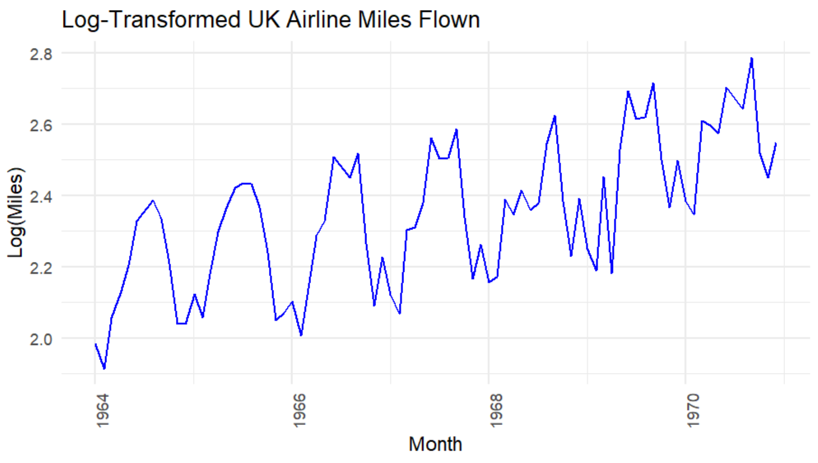
Explanation of some code:

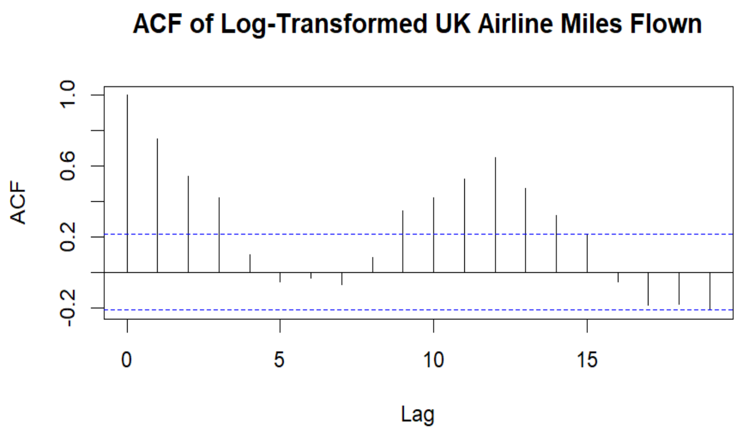


```r
# Take the natural logarithm of the 'Miles' data
df$log_Miles <- log(df$`Miles(in millions)`)  # Corrected

# Check for NA values in the 'log_Miles' column
sum(is.na(df$log_Miles))

# Remove rows with missing values in 'log_Miles'
df_clean <- df[complete.cases(df$log_Miles), ]

# Plot the log-transformed time series
ggplot(df_clean, aes(x = Month, y = log_Miles)) +
  geom_line(color = "blue") +
  labs(title = "Log-Transformed UK Airline Miles Flown",
       x = "Month",
       y = "Log(Miles)") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))  # Rotate x labels for better read

# Calculate and plot the Autocorrelation Function (ACF)
acf(df_clean$log_Miles, main = "ACF of Log-Transformed UK Airline Miles Flown")
```

The logarithmic transformation of the UK airline miles flown data is carried out by this R code to stabilize variance and display the transformed time series. The `Miles (in millions)} column is first subjected to the natural logarithm, and the result is then stored in a new column named `log_Miles}. Next, to make sure the transformation is successful, the code uses the `sum(is.na())} function to check for any missing (NA) values in the {log_Miles` column. The `complete.cases()} function is then used to eliminate missing value rows, resulting in the cleaned dataset `df_clean}, which only includes valid log-transformed values. Following data cleaning, **ggplot2** is used to plot the log-transformed time series, with the log-transformed miles flown ({log_Miles}) represented on the y-axis and time ({Month}) on the x-axis.

Plot



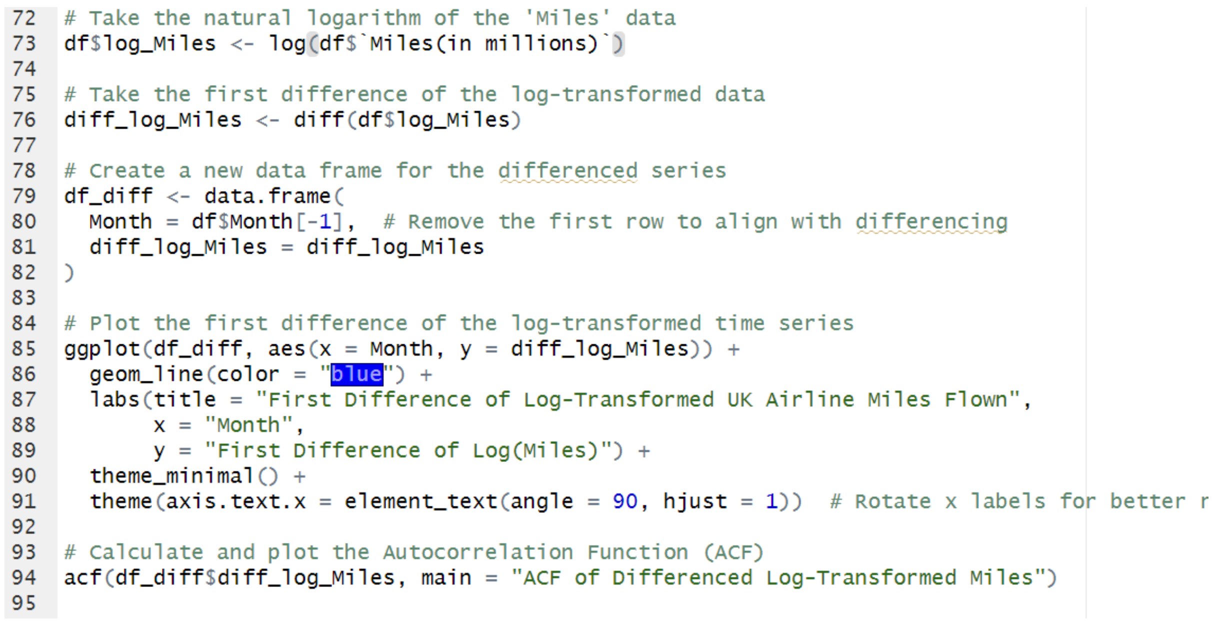


a. The log transformation lessens large variations and stabilizes the variation of the UK airline miles flown time series. Larger numbers are compressed, making the data less skewed and more linear. The smoother trend over time demonstrates how valuable this modification is for data with increasing patterns. Overall, by reducing the influence of extreme results, the log transformation makes analysis more understandable.

b. Relationships between the log-transformed UK airline mile flow and its lags are displayed in the sample autocorrelation function (ACF) plot. The ACF shows a substantial positive correlation at lag 1, which progressively declines over the following lags. A significant pattern that appears periodically indicates that the data may have some seasonality or periodic structure. The autocorrelations increase again at lags of 10 to 12. The time series has some autocorrelation up to about lag 15, according to the statistically significant ACF values from outside the blue dashed lines.

3. Take the first difference of the natural logarithm of the data and plot this new time series.
   1. What impact has the log transformation had on the time series?
   2. Find the autocorrelation function for this time series.
   3. Interpret the sample autocorrelation function.

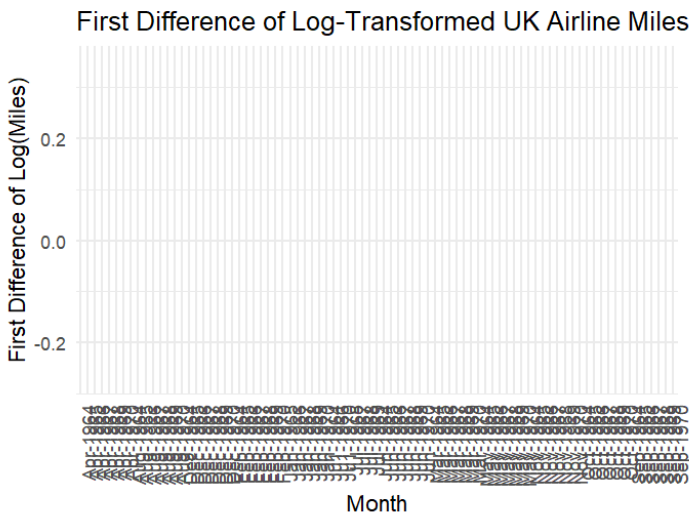
Explanation of some code:

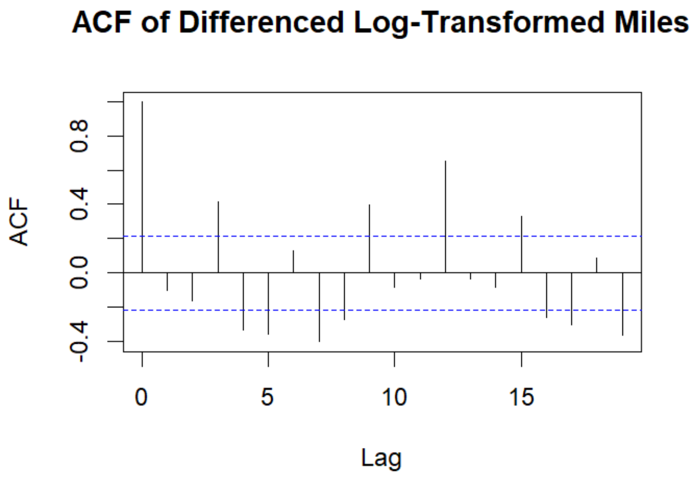


```r
# Take the natural logarithm of the 'Miles' data
df$log_Miles <- log(df$`Miles(in millions)`)

# Take the first difference of the log-transformed data
diff_log_Miles <- diff(df$log_Miles)

# Create a new data frame for the differenced series
df_diff <- data.frame(
  Month = df$Month[-1],  # Remove the first row to align with differencing
  diff_log_Miles = diff_log_Miles
)

# Plot the first difference of the log-transformed time series
ggplot(df_diff, aes(x = Month, y = diff_log_Miles)) +
  geom_line(color = "blue") +
  labs(title = "First Difference of Log-Transformed UK Airline Miles Flown",
       x = "Month",
       y = "First Difference of Log(Miles)") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))  # Rotate x labels for better r

# Calculate and plot the Autocorrelation Function (ACF)
acf(df_diff$diff_log_Miles, main = "ACF of Differenced Log-Transformed Miles")
```

The `Miles (in millions)` column's natural logarithm is first calculated in this R code to transform the UK airline miles flown data and help stabilize the variance. Next, it uses `diff()` to compute the first difference of the log-transformed data, which helps eliminate trends and stabilize the time series. To ensure that the {Month} column is correctly aligned following differencing, a new data frame called `df_diff} is created and used to store the differenced series. The autocorrelation function (ACF) of the first-differenced log-transformed series is then computed and plotted by the code to help identify seasonality or other correlations by analyzing patterns or dependencies between subsequent time points.

Plot



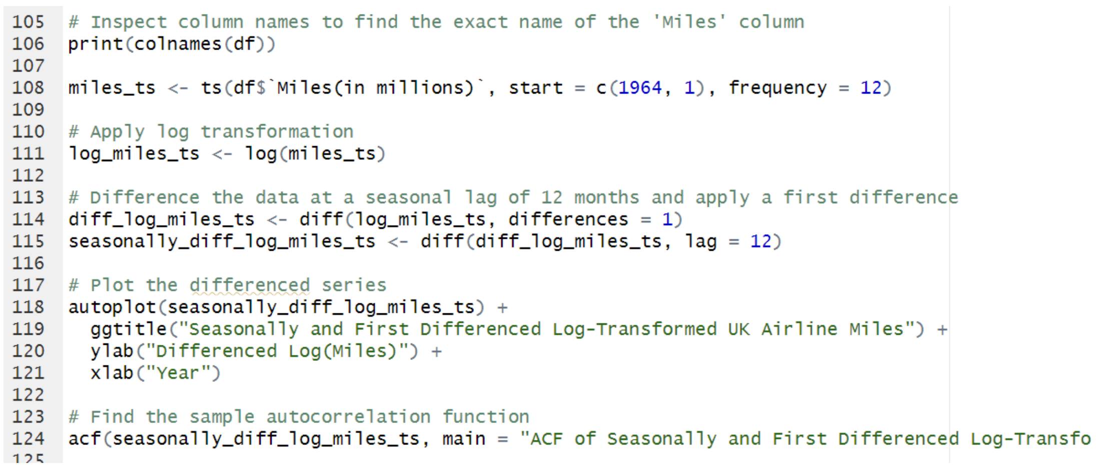


a. The UK airline miles flow time series' variance has been stabilized by the log transformation, making it less dependent on shifts that can mask data patterns. The plot's particularly compressed range on the y-axis, however, suggests that even though the transformation has lessened variance, seasonal effects and trends might not have completely disappeared. The time series has become more stationary, as indicated by the horizontal lines close to zero, which implies that the difference between the series fluctuates around a mean. However, the plot's lack of discernible data points indicates that additional differentiation or smoothing is required to completely reveal the underlying pattern.

b. Some autocorrelation has been eliminated by differencing, as evidenced by the large spike in the ACF for the differenced log-transformed miles at lag 1. The autocorrelation rapidly decreases after this, indicating enhanced stationarity. On the other hand, significant autocorrelations near lag 10 might indicate lingering seasonality. Since most autocorrelation values lie between the blue dashed lines, they are likely not statistically important. With some possible residual autocorrelation from seasonality, the ACF generally indicates that the data set is relatively stationary.

4. Difference the data at a season lag of 12 months and also apply a first difference to the data. Plot the differenced series.
   1. What effect has the differencing had on the time series?
   2. Find the sample autocorrelation function.
   3. What does the sample autocorrelation function tell you about the behavior of the differenced series?

Explanation of some code:

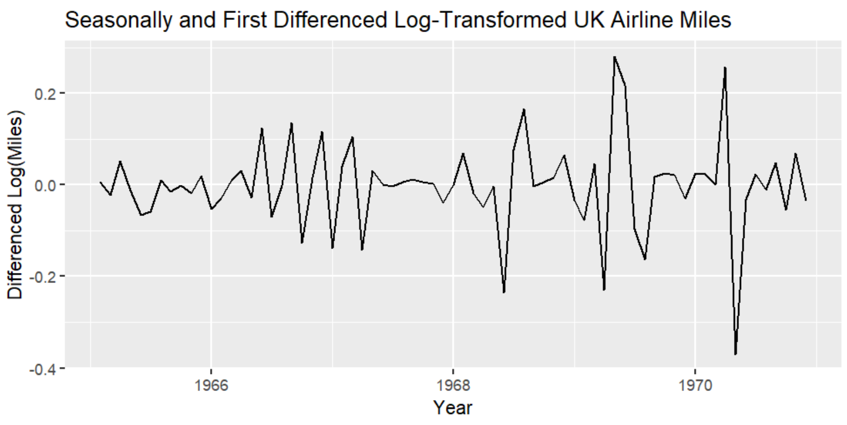


```
# Inspect column names to find the exact name of the 'Miles' column
print(colnames(df))

miles_ts <- ts(df$`Miles(in millions)`, start = c(1964, 1), frequency = 12)

# Apply log transformation
log_miles_ts <- log(miles_ts)

# Difference the data at a seasonal lag of 12 months and apply a first difference
diff_log_miles_ts <- diff(log_miles_ts, differences = 1)
seasonally_diff_log_miles_ts <- diff(diff_log_miles_ts, lag = 12)

# Plot the differenced series
autoplot(seasonally_diff_log_miles_ts) +
  ggtitle("Seasonally and First Differenced Log-Transformed UK Airline Miles") +
  ylab("Differenced Log(Miles)") +
  xlab("Year")

# Find the sample autocorrelation function
acf(seasonally_diff_log_miles_ts, main = "ACF of Seasonally and First Differenced Log-Transfo
```

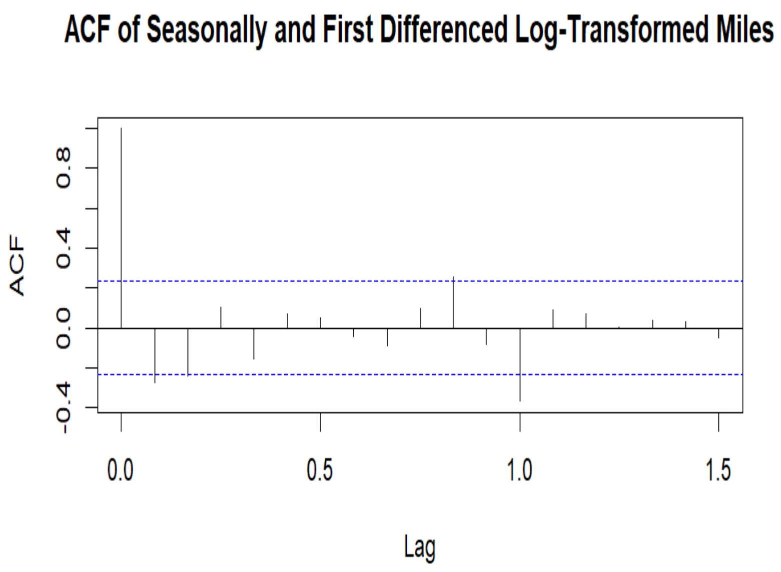
To make a time series dataset of UK airline miles flown more stationary and appropriate for additional analysis, this R code applies transformations and differencing to the data.
To guarantee proper column referencing, print(colnames(df)) is used to first determine the precise name of the "Miles" column. Next, starting in January 1964 and with a frequency of 12, the Miles(in millions) column is transformed into a time series object (miles_ts), which represents monthly data. After the series' variance is stabilized through log transformation, trends and seasonality are eliminated by applying a first difference and seasonal difference, with a 12-month lag.

To analyze the connections between lag observations, the autocorrelation function (ACF) is calculated and the transformed series is plotted using autoplot(). Reduced autocorrelation in the ACF plot suggests that the series is now more stationary and has characteristics similar to white noise. Lastly, the code offers written interpretations, stating that the series has been effectively fixed, and trends eliminated through differencing, and the ACF plot indicates that there is little autocorrelation, which suggests that the information no longer displays strong seasonal trends.

Plot



The original series' variance had effectively stabilized, the trend was eliminated, and the series became stationary with the help of the initial differentiation and seasonal differences applied at a 12-month lag. With fewer severe peaks and an oscillation around zero, the differenced series still exhibits some volatility. It seems that seasonal patterns have vanished, but there are still brief variations. There are some discernible irregularities, with apparent spikes around 1968 and 1970.

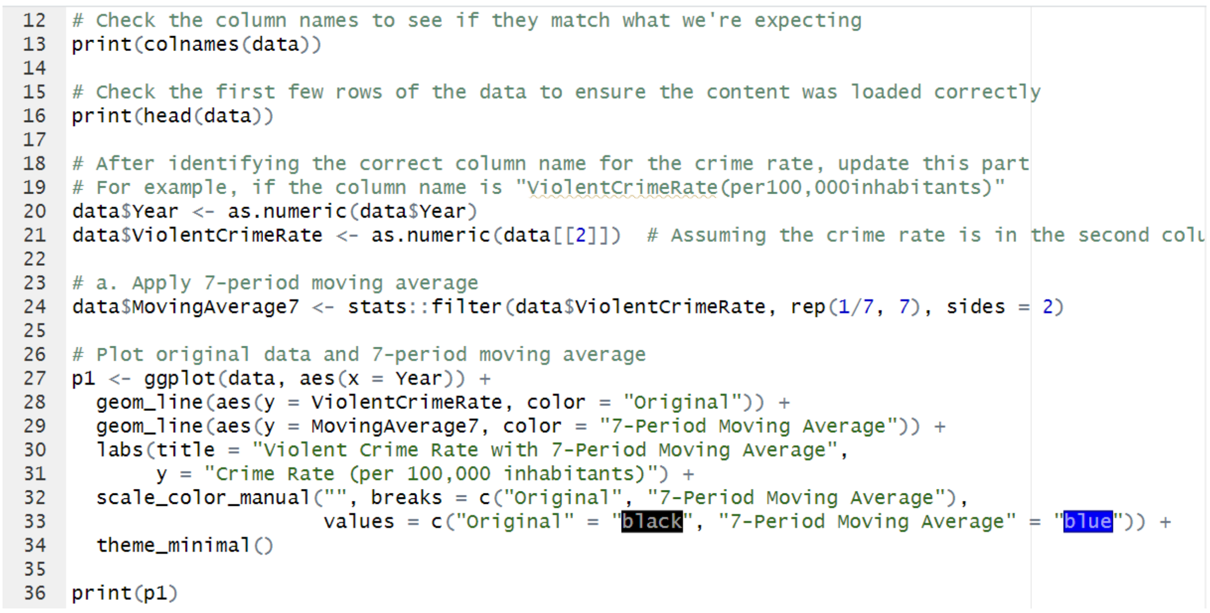


After lag 0, the results in the differenced series' autocorrelation function (ACF) fall within the range of confidence bands, suggesting that insignificant autocorrelations are still present. This implies that the data's autocorrelation has been successfully eliminated by the differencing. The absence of obvious seasonal autocorrelation in the ACF plot indicates that seasonal trends have mostly disappeared. At this point, the series behaves like a white disturbances process.

U.S. National Violent Crime Rate

2. Consider the violent crime rate data table.
   1. Use a seven-period moving average to smooth the data. Plot both the smoothed data and the original readings on the same axes.
   2. What has the moving average done?
   3. Repeat the procedure with a three-period moving average.
   4. What is the effect of changing the span of the moving average?

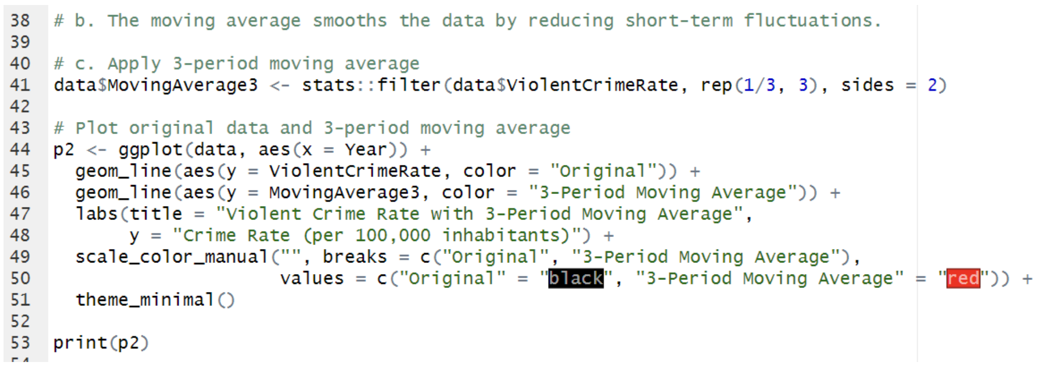
Explanation of some code:



```r
# Check the column names to see if they match what we're expecting
print(colnames(data))

# Check the first few rows of the data to ensure the content was loaded correctly
print(head(data))

# After identifying the correct column name for the crime rate, update this part
# For example, if the column name is "ViolentCrimeRate(per100,000inhabitants)"
data$Year <- as.numeric(data$Year)
data$ViolentCrimeRate <- as.numeric(data[[2]])  # Assuming the crime rate is in the second colu

# a. Apply 7-period moving average
data$MovingAverage7 <- stats::filter(data$ViolentCrimeRate, rep(1/7, 7), sides = 2)

# Plot original data and 7-period moving average
p1 <- ggplot(data, aes(x = Year)) +
  geom_line(aes(y = ViolentCrimeRate, color = "Original")) +
  geom_line(aes(y = MovingAverage7, color = "7-Period Moving Average")) +
  labs(title = "Violent Crime Rate with 7-Period Moving Average",
       y = "Crime Rate (per 100,000 inhabitants)") +
  scale_color_manual("", breaks = c("Original", "7-Period Moving Average"),
                     values = c("Original" = "black", "7-Period Moving Average" = "blue")) +
  theme_minimal()

print(p1)
```

This R script loads and analyzes the crime rate data, smoothing variations in the number of violent crimes with a 7-period moving average and producing an output visualization. Initially, it verifies that the data is loaded correctly by checking the column names ({colnames()}) and printing the first few rows (`head()}). A numeric format is created by identifying and converting the violent crime rate column. The **stats** package's `filter()` function is used to apply a 7-period moving average. The parameter {rep(1/7, 7)} generates a moving window of seven values, which are then averaged to reduce short-term fluctuations.

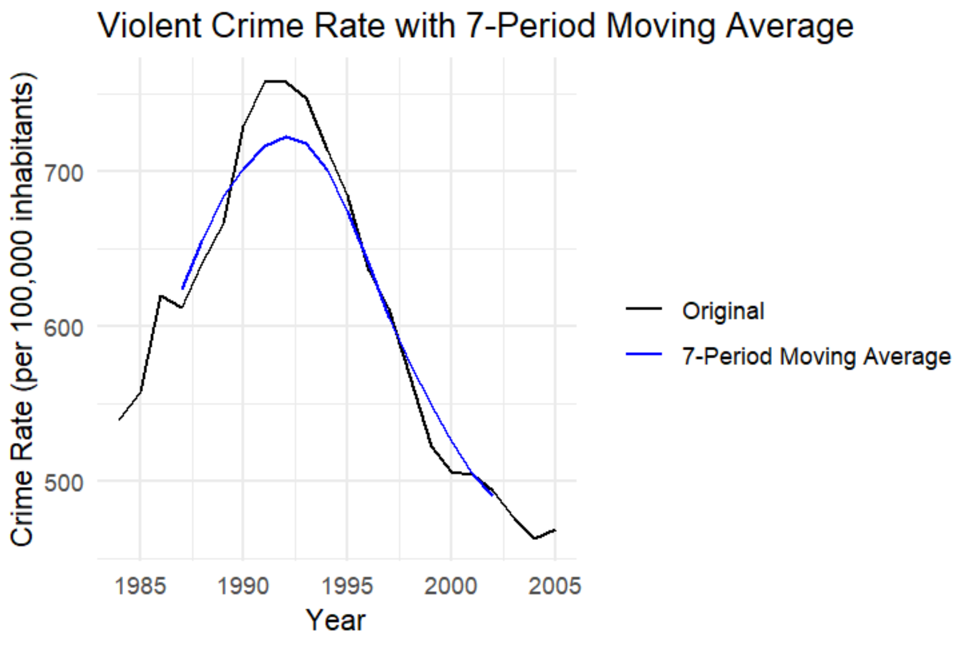
Next, using ggplot2, the code displays the smoothed series as well as the initial violent crime rate. Plotted are two lines: the 7-period average movement in blue and the original crime rate in black. It is now simpler to observe long-term variations in the crime rate due to this visualization's ability to highlight the overall trend and reduce noise from short-term variability in the data.



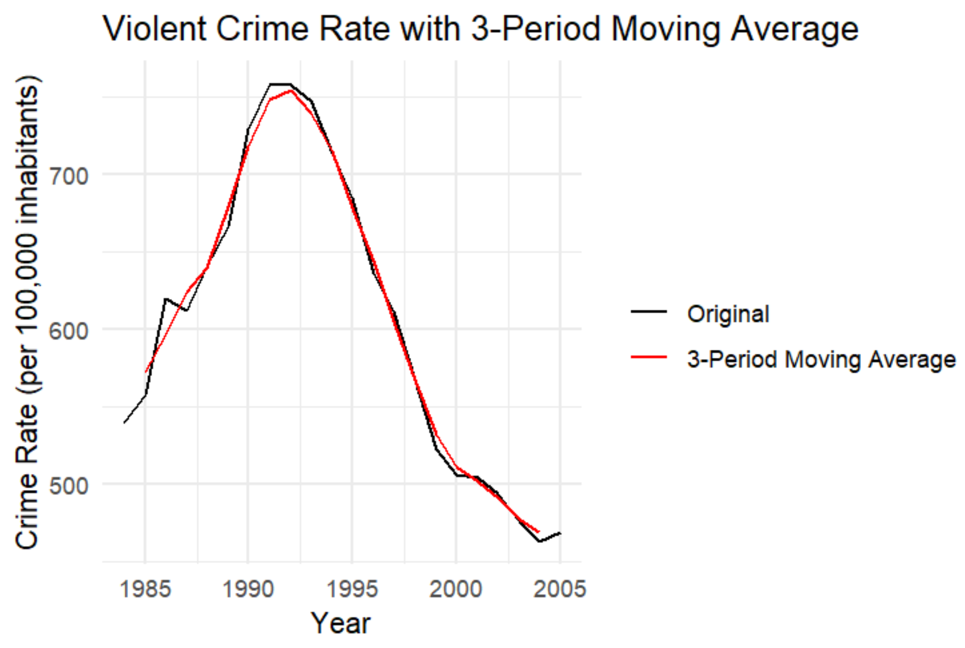
```r
# b. The moving average smooths the data by reducing short-term fluctuations.

# c. Apply 3-period moving average
data$MovingAverage3 <- stats::filter(data$ViolentCrimeRate, rep(1/3, 3), sides = 2)

# Plot original data and 3-period moving average
p2 <- ggplot(data, aes(x = Year)) +
  geom_line(aes(y = ViolentCrimeRate, color = "Original")) +
  geom_line(aes(y = MovingAverage3, color = "3-Period Moving Average")) +
  labs(title = "Violent Crime Rate with 3-Period Moving Average",
       y = "Crime Rate (per 100,000 inhabitants)") +
  scale_color_manual("", breaks = c("Original", "3-Period Moving Average"),
                     values = c("Original" = "black", "3-Period Moving Average" = "red")) +
  theme_minimal()

print(p2)
```

This R code smooths the data to lessen short-term shifts by applying a 3-period average movement to the rate of violent crimes data and plotting the results. The moving average is applied using the stats package's filter() function, which creates a window that slides in and out of three figures to calculate the average. This method smoothes out noise in the data over less time than the 7-period moving average that was previously used. The initial criminal activity rate (in black), as well as the 3-period moving average (in red), are both displayed in the plot, which was made with ggplot2. While allowing for small fluctuations over shorter periods, the moving average line emphasizes the overall pattern, making it simpler to spot broad trends.

Plot



It is now simpler to see the general trend in the number of violent crimes since the moving average has eliminated short-term swings. The 7-period moving average (in blue) shows a more gradual curve, which reduces noise, compared to the original data (in black), which shows more variability and abrupt changes. The rate of crime generally showed an upward trend until the early 1990s, at which point it steadily declined. Without the smaller changes, the moving average makes the highest point and fall of the crime rate more evident. Additionally, by drawing attention to the crime rate's wider pattern, a moving average helps to shed light on key moments, particularly those surrounding the peak in the early 1990s. It provides a more reliable depiction of the long-term trend of the data by reducing short-term spikes and dips.

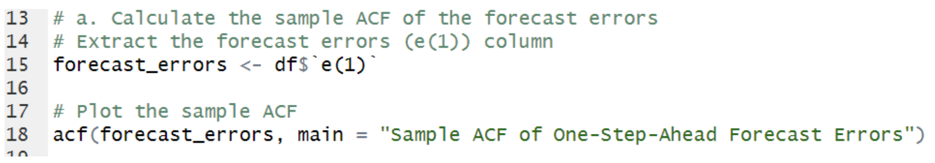


A seven-period moving average offers more smoothing than a three-period moving average. More short-term fluctuations are captured by the red line, which is the three-period moving average, as it closely tracks the initial black line. Because of its shorter span, the valleys and peaks are less smoothed and more pointed, responding to changes in the data faster. Overall, the three-period average movement provides a reasonable middle ground between highlighting trends and preserving a portion of the unique data's variation.

One-Step-Ahead Forecast Errors

3. Consider the table that contains 40 one-step-ahead forecast errors from a forecasting model.
    1. Find the sample ACF of the forecast errors. Interpret the results.
    2. Construct a normal probability plot of the forecast errors.

Explanation of some code:



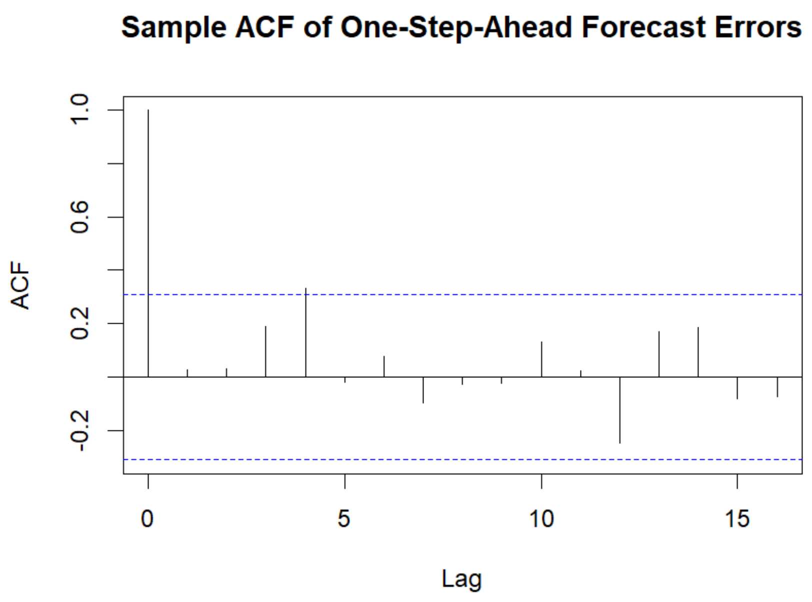
```
# a. Calculate the sample ACF of the forecast errors
# Extract the forecast errors (e(1)) column
forecast_errors <- df$`e(1)`

# Plot the sample ACF
acf(forecast_errors, main = "Sample ACF of One-Step-Ahead Forecast Errors")
```

Using the forecast and ggplot2 libraries, this R code computes and displays a sample autocorrelation function (ACF) of one-step-ahead forecast errors. Initially, the forecast errors are taken out of the dataset's {e(1)} column and placed in the `forecast_errors} variable. Next, the ACF is computed and visualized using the `acf()` function, showing the correlations among the anticipated errors at various time lags. With essential correlations suggesting a not well-fitted model for forecasting and insufficient autocorrelations suggesting that the predictive model's residuals are random, the ACF plot aids in determining whether the forecast mistakes are distinct over time.

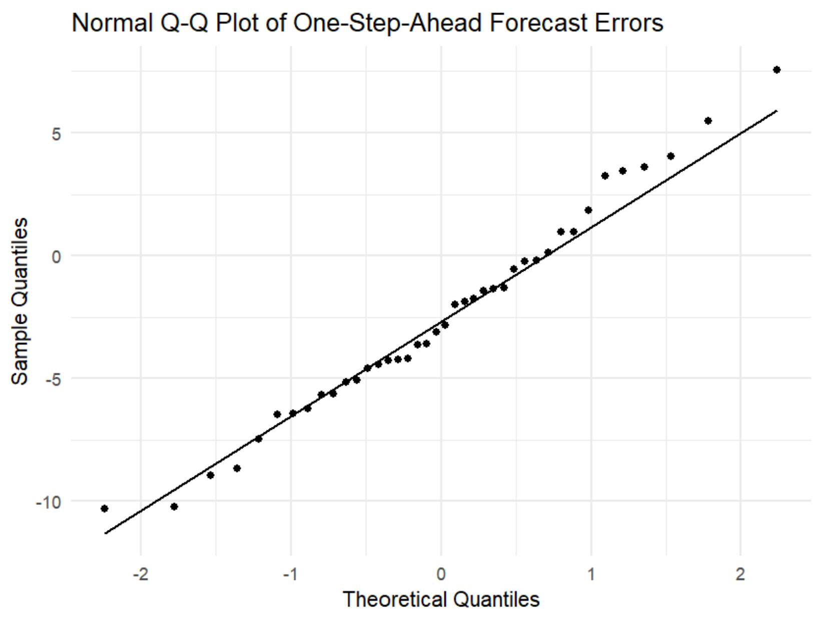
```r
# b. Construct a normal probability plot of the forecast errors
ggplot(df, aes(sample = forecast_errors)) +
  stat_qq() +
  stat_qq_line() +
  labs(title = "Normal Q-Q Plot of One-Step-Ahead Forecast Errors",
       x = "Theoretical Quantiles",
       y = "Sample Quantiles") +
  theme_minimal()
```

Using ggplot2, this R code creates a normal probability plot (Q-Q plot) for the forecast errors. The Q-Q plot is produced by the `stat_qq()` function, which contrasts the forecast errors' distribution with a conceptual normal distribution. The function `stat_qq_line()` adds a line. The plot displays theoretical versus sample quantiles, as indicated by the labels on the axes and title. For improved readability, the `theme_minimal()` operates and applies a simple, minimalist theme to the graph.

Plot

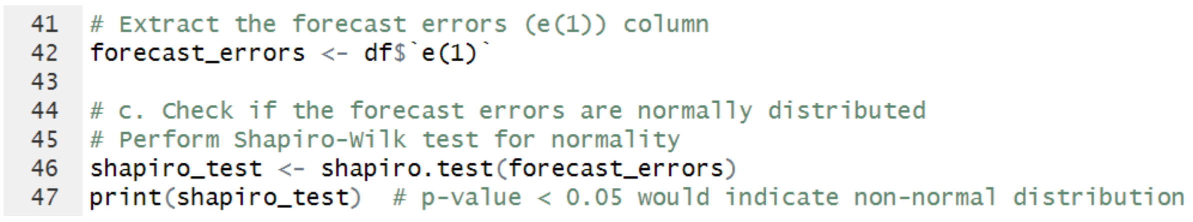


The plot displays the one-step-ahead forecast errors' autocorrelation function (ACF), with most autocorrelations falling within the blue significance boundaries. A substantial positive autocorrelation appears in the first lag, indicating the possibility of a serial relationship in the errors at this lag. The fact that the remaining lags all fall inside the significance bounds suggests that they are like zero. The model may fail to eliminate all temporal correlations from the forecast errors if there is a significant autocorrelation at lag 1.



The quantiles that represent the forecast errors are compared to the normal distribution, in theory, using the Q-Q plot. Some of the points are in alignment with the 45-degree line, but the bottom and upper extremes of the tails diverge, suggesting that the range of forecast errors is not normal. The lower quantiles (below -2) show a noticeable deviation, indicating the presence of extremely negative forecast errors. The residuals might not exactly comply with a normal distribution, which might prove problematic for model assumptions based on normally distributed errors, as indicated by this departure from normalcy.

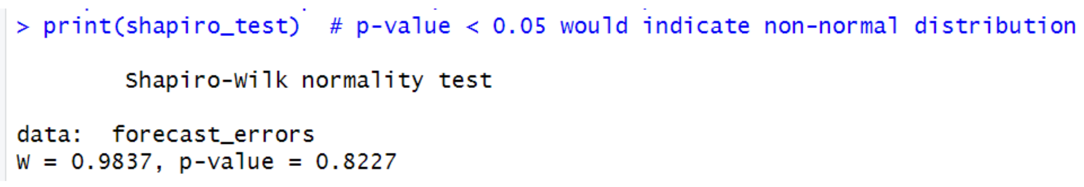
Continuation of code:



```
# Extract the forecast errors (e(1)) column
forecast_errors <- df$`e(1)`

# c. Check if the forecast errors are normally distributed
# Perform Shapiro-Wilk test for normality
shapiro_test <- shapiro.test(forecast_errors)
print(shapiro_test)  # p-value < 0.05 would indicate non-normal distribution
```

The one-step-ahead forecast errors are represented by column e(1) in the dataset df, which is extracted by this code. Next, the Shapiro-Wilk test is used to determine whether these forecast errors have a normal distribution. The Shapiro returns two important values.test() function: the p-value, which aids the calculation of normality, and W, the result of the test statistic.

Result of Shapiro-Wilk Test



```
> print(shapiro_test)  # p-value < 0.05 would indicate non-normal distribution

        Shapiro-Wilk normality test

data:  forecast_errors
W = 0.9837, p-value = 0.8227
```

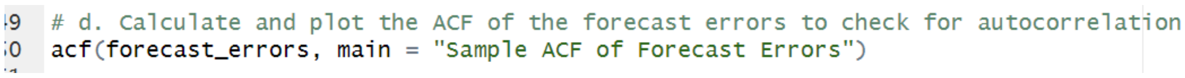
The test statistic that shows how closely the sample fits into a normal distribution is W = 0.9837.

Since the p-value of 0.8227 is much higher than 0.05, the null hypothesis—that is, the data is normally distributed—cannot be ruled out.

Interpretation

No proof would indicate that the errors in forecasting deviate from normality because the p-value is much larger than 0.05. As a result, the errors in forecasting are normally distributed.

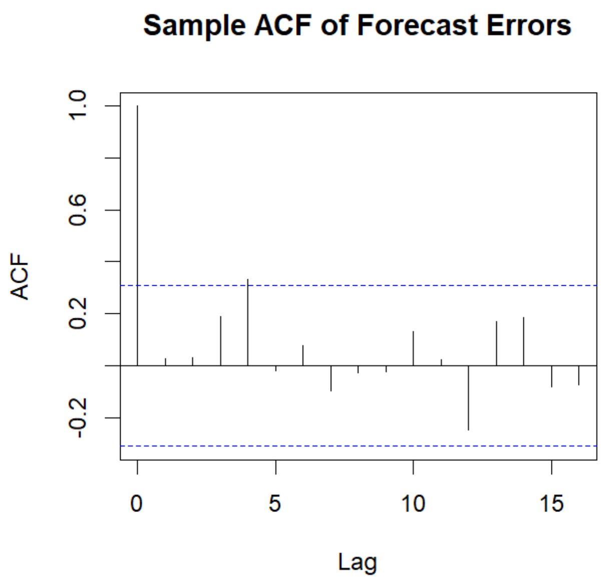
Code:



```r
# d. Calculate and plot the ACF of the forecast errors to check for autocorrelation
acf(forecast_errors, main = "Sample ACF of Forecast Errors")
```

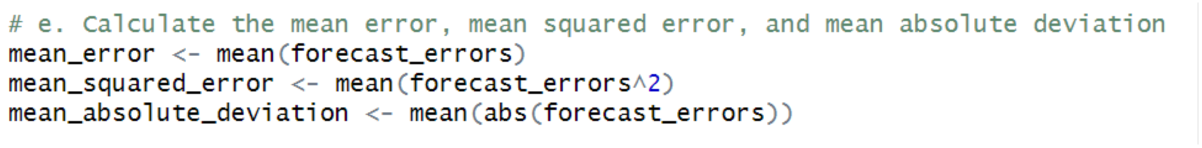
The autocorrelation function (ACF) of the forecast errors is computed and plotted on this line. The residuals, or errors, of the forecasts can be analyzed using the ACF to find any patterns or correlations. The autocorrelation function of a time series can be calculated and plotted using the R function acf(). The "forecast_errors" vector represents the forecast errors, or the discrepancies between the expected and actual values.

Plot



The sample autocorrelation function (ACF) of forecast errors at various lags is displayed in this plot. The forecast errors at lag one is repeatedly correlated, as shown by the first lag's substantial positive autocorrelation. On the contrary, autocorrelations at higher lags appear not to be statistically significant, since all other lags fall within the range of confidence bounds. Given the circumstances, the model appears to work well at removing autocorrelation for later lags, but it may have some trouble doing so at lag one.

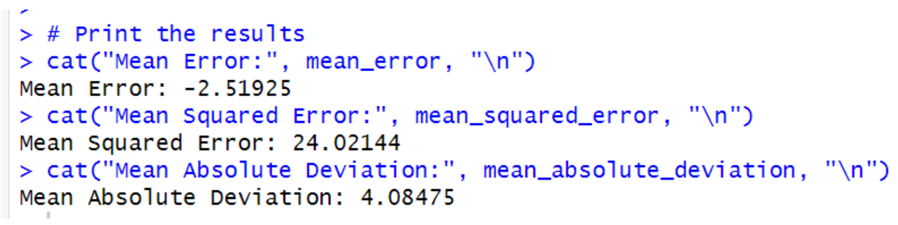
Code:



```r
# e. Calculate the mean error, mean squared error, and mean absolute deviation
mean_error <- mean(forecast_errors)
mean_squared_error <- mean(forecast_errors^2)
mean_absolute_deviation <- mean(abs(forecast_errors))
```

To evaluate the precision of a set of forecast errors, this code computes three important metrics. Using mean(forecast_errors), which provides the average of the forecast errors and shows any bias within the estimates, the mean error is calculated first. Next, using mean(forecast_errors^2) to compute the mean of the squared forecast errors, the mean squared error (MSE) is computed. This indicates the variability in the errors and penalizes larger discrepancies more severely. Lastly, mean(abs(forecast_errors)) yields the mean absolute deviation (MAD), which provides a clear picture of the typical forecast error size by displaying the average magnitude of the mistakes without considering their direction.

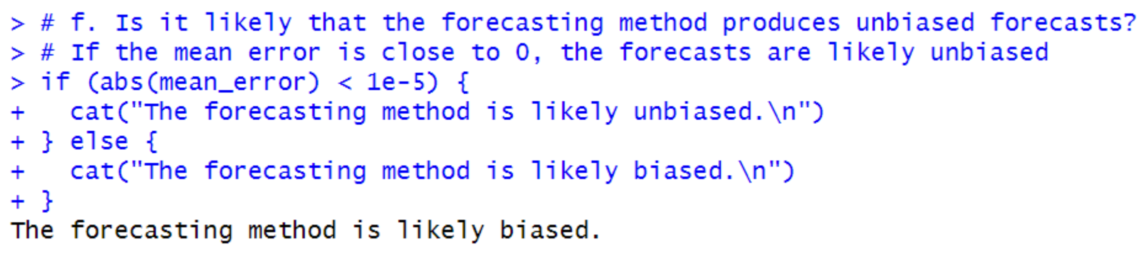
Result



```
> # Print the results
> cat("Mean Error:", mean_error, "\n")
Mean Error: -2.51925
> cat("Mean Squared Error:", mean_squared_error, "\n")
Mean Squared Error: 24.02144
> cat("Mean Absolute Deviation:", mean_absolute_deviation, "\n")
Mean Absolute Deviation: 4.08475
```

Based on the mean error, the forecasts frequently understate the actual values by about 2.52 units on average, according to the results. With larger errors impacted more heavily, the mean squared error of 24.02 indicates that there is substantial variation in the forecast errors. The forecast errors typically have a magnitude of 4.08 units, as indicated by the mean absolute deviation of 4.08, which gives a clear indication of the average error size.

Code and Result:



```
> # f. Is it likely that the forecasting method produces unbiased forecasts?
> # If the mean error is close to 0, the forecasts are likely unbiased
> if (abs(mean_error) < 1e-5) {
+   cat("The forecasting method is likely unbiased.\n")
+ } else {
+   cat("The forecasting method is likely biased.\n")
+ }
The forecasting method is likely biased.
```

The absolute value of the mean error is compared to an exceedingly small threshold ({1e-5}) by the code to determine whether the forecasting approach is likely unbiased. The forecasts are regarded as possibly unbiased if the mean error is less than this limit; if not, they are regarded as biased. The outcome in this instance suggests that the method used for forecasting is biased, implying that there is a systematic ability in the forecasts to underestimate or overstate the actual values and that the average forecast errors are not near enough to zero.