Predictive Modeling with Time Series Data

Felipe O. Cerezer

2024-08-15

1. Required Libraries

First, ensure you have the required libraries installed and load them into your R environment.

Install the packages if not already installed

install.packages(c("forecast", "prophet", "ggplot2"))

1.1 Load necessary libraries

```
library(forecast) # For ARIMA modeling
```

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

```
library(prophet) # For Prophet modeling
```

```
## Loading required package: Rcpp
```

```
## Loading required package: rlang
```

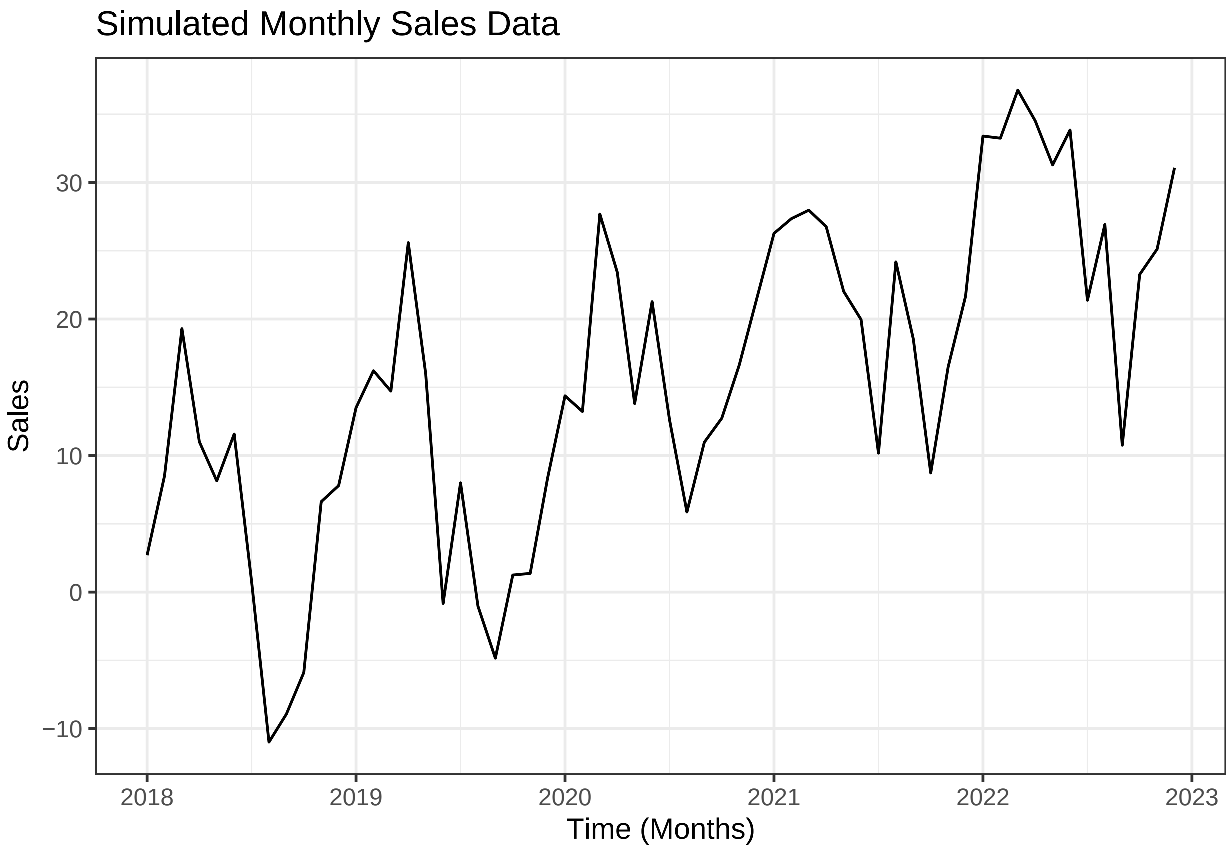
```
library(ggplot2) # For data visualization
```

2. Simulate Fake Time Series Data

I simulate a time series representing monthly sales data over five years. The data includes a trend, seasonality (e.g., summer and winter peaks in tourism), and random noise (unpredictable part of the data).

```
# Set seed for reproducibility
set.seed(123)

# Simulate Fake Time Series Data
n <- 60  # 60 months (5 years)
time <- seq(1, n)
seasonality <- 10 * sin(2 * pi * time / 12)  # Seasonal component
trend <- 0.5 * time  # Trend component
noise <- rnorm(n, mean = 0, sd = 5)  # Random noise
sales <- trend + seasonality + noise  # Combine components

# Create a time series object
sales_ts <- ts(sales, start = c(2018, 1), frequency = 12)
```

```r
# Plot the simulated sales data
autoplot(sales_ts) +
  ggtitle("Simulated Monthly Sales Data") +
  ylab("Sales") + xlab("Time (Months)")+
  theme_bw()
```



Explanation:

- I simulate monthly sales data over five years. The data contains three components: a trend, a seasonal pattern, and random noise.

- sales_ts is a time series object starting from January 2018, with a frequency of 12 (monthly data).

3. ARIMA Modeling and Forecasting

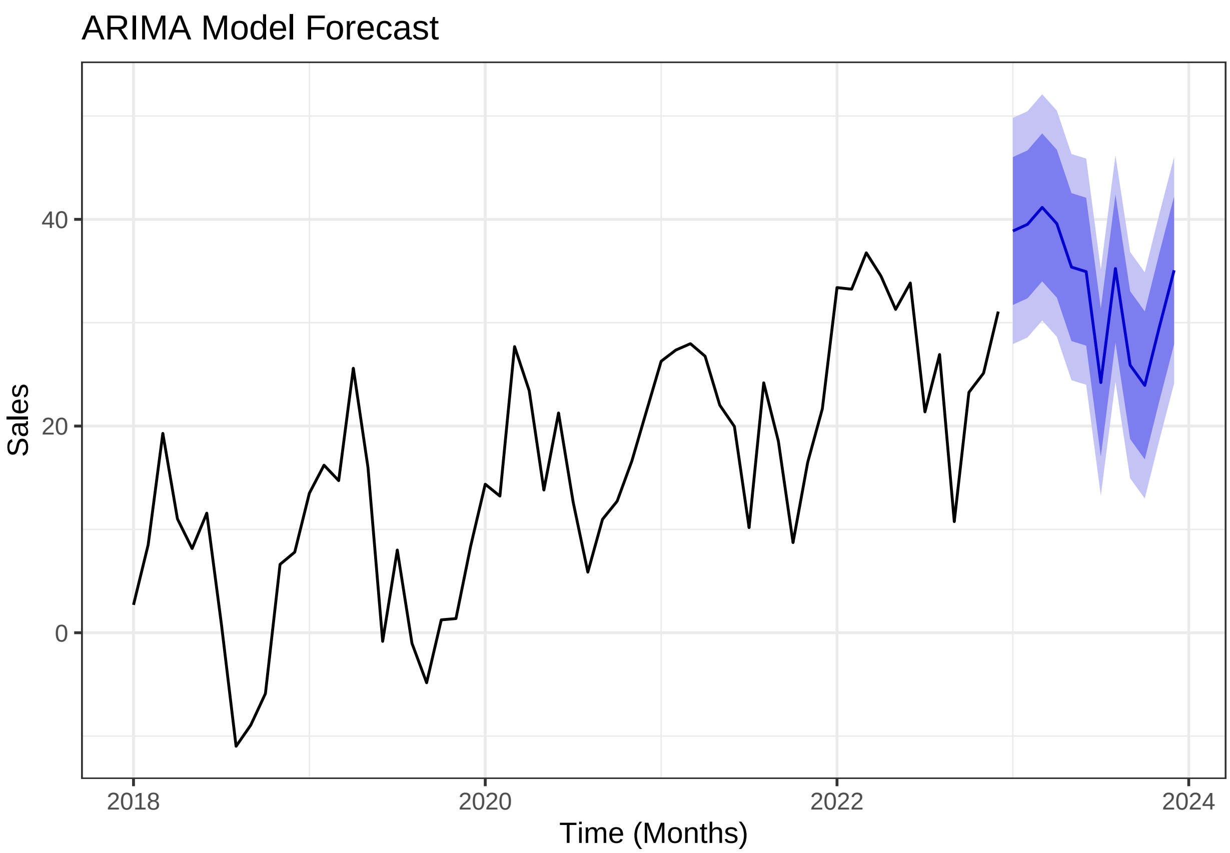
The auto.arima function is used to fit the ARIMA model, which is then used to forecast future values

```r
# ARIMA Modeling
fit_arima <- auto.arima(sales_ts)  # Automatically fit ARIMA model
summary(fit_arima)  # Display model details
```

```
## Series: sales_ts
## ARIMA(0,0,0)(1,1,0)[12] with drift
##
## Coefficients:
##          sar1   drift
##       -0.6482  0.5105
```

```
## s.e.   0.1053  0.0445
##
## sigma^2 = 31.18:  log likelihood = -152.91
## AIC=311.82   AICc=312.37   BIC=317.43
##
## Training set error measures:
##                      ME     RMSE      MAE      MPE     MAPE      MASE
## Training set -0.1763034 4.889015 3.344342 12.61809 63.78359 0.4208828
##                     ACF1
## Training set -0.05025177
```

```r
# Forecast the next 12 months using ARIMA
forecast_arima <- forecast(fit_arima, h = 12)

# Plot ARIMA forecast
autoplot(forecast_arima) +
  ggtitle("ARIMA Model Forecast") +
  ylab("Sales") + xlab("Time (Months)")+
  theme_bw()
```



Explanation:

- auto.arima automatically identifies the best ARIMA model based on the time series data.

- The model is used to forecast the next 12 months, and the forecast is plotted with confidence intervals.

4. Prophet Modeling and Forecasting

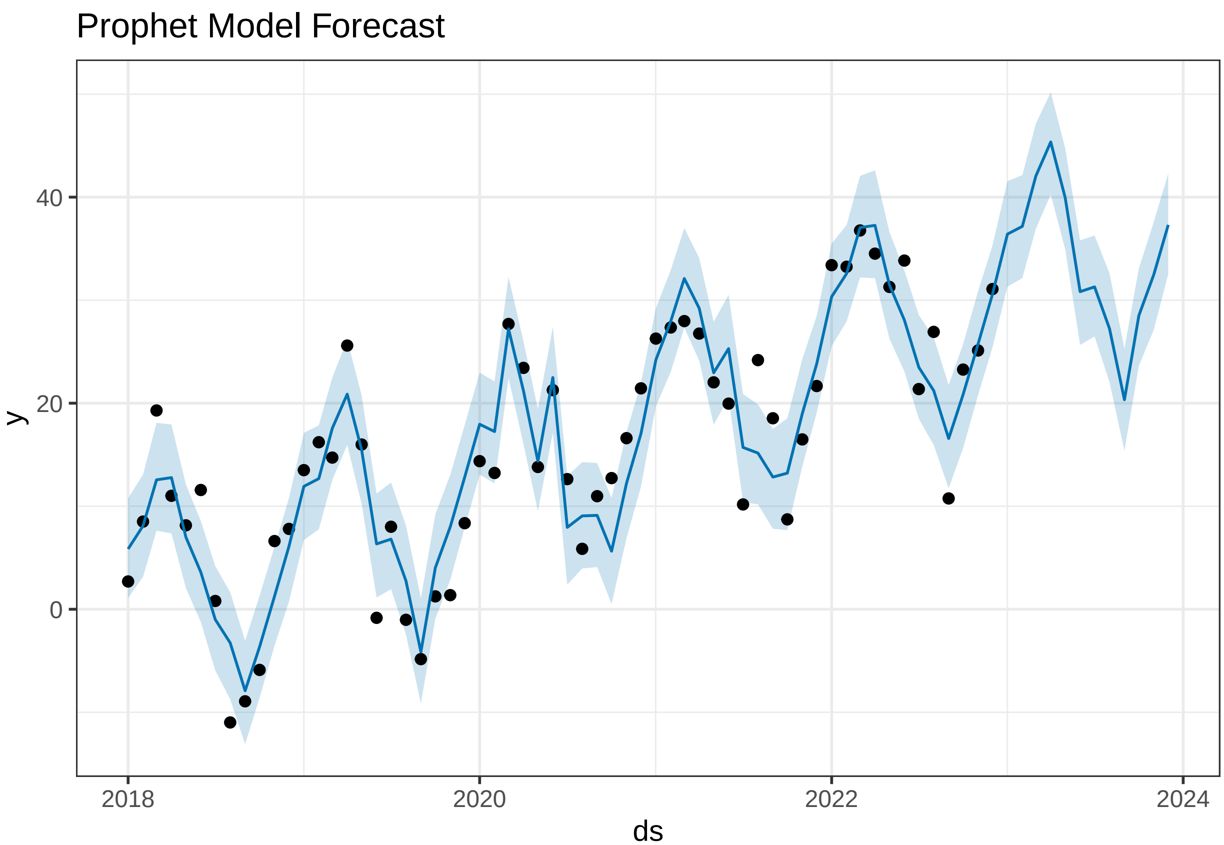
I fit a Prophet model, a robust tool for time series forecasting, especially useful for data with strong seasonal patterns.

```
# Prepare data for Prophet
sales_df <- data.frame(ds = seq.Date(from = as.Date("2018-01-01"),
                                     by = "month", length.out = n),
                       y = as.numeric(sales_ts))

# Fit the Prophet model
model_prophet <- prophet(sales_df)
```

```
## Disabling weekly seasonality. Run prophet with weekly.seasonality=TRUE to override this.
```

```
## Disabling daily seasonality. Run prophet with daily.seasonality=TRUE to override this.
```

```
# Create future dataframe for forecasting
future <- make_future_dataframe(model_prophet, periods = 12, freq = "month")
forecast_prophet <- predict(model_prophet, future)

# Plot Prophet forecast
plot(model_prophet, forecast_prophet) +
  ggtitle("Prophet Model Forecast")+
  theme_bw()
```



Explanation:

- The data is reformatted to be compatible with Prophet.

- The prophet function fits the model, and forecast the next 12 months.

- The forecasted values and their uncertainty intervals are plotted.

5. Model Validation with ARIMA

To evaluate the model's performance, I split the data into training and test sets, fit the model on the training data, and assess its accuracy on the test data.

```
# Split data into training (first 4 years) and test (last year)
train_ts <- window(sales_ts, end = c(2021, 12))  # Training: 2018-2021
test_ts <- window(sales_ts, start = c(2022, 1))  # Test: 2022 onwards

# Fit ARIMA on the training set and forecast on the test set
fit_arima_train <- auto.arima(train_ts)
forecast_arima_test <- forecast(fit_arima_train, h = length(test_ts))

# Calculate and print accuracy metrics
accuracy_metrics <- accuracy(forecast_arima_test, test_ts)
print(accuracy_metrics)
```

```
##                      ME     RMSE      MAE       MPE     MAPE      MASE
## Training set -0.2079846 4.860134 3.283365 17.138782 73.24193 0.4307880
## Test set      0.7428376 4.842629 3.448885 -4.049777 17.75497 0.4525047
##                      ACF1 Theil's U
## Training set -0.009731465        NA
## Test set     -0.221902068 0.4274927
```

Explanation:

- I split the time series into a training set (2018-2021) and a test set (2022).

- The ARIMA model is trained on the training set and used to forecast the test set.

- accuracy computes the accuracy of the forecast, including metrics like RMSE and MAE.

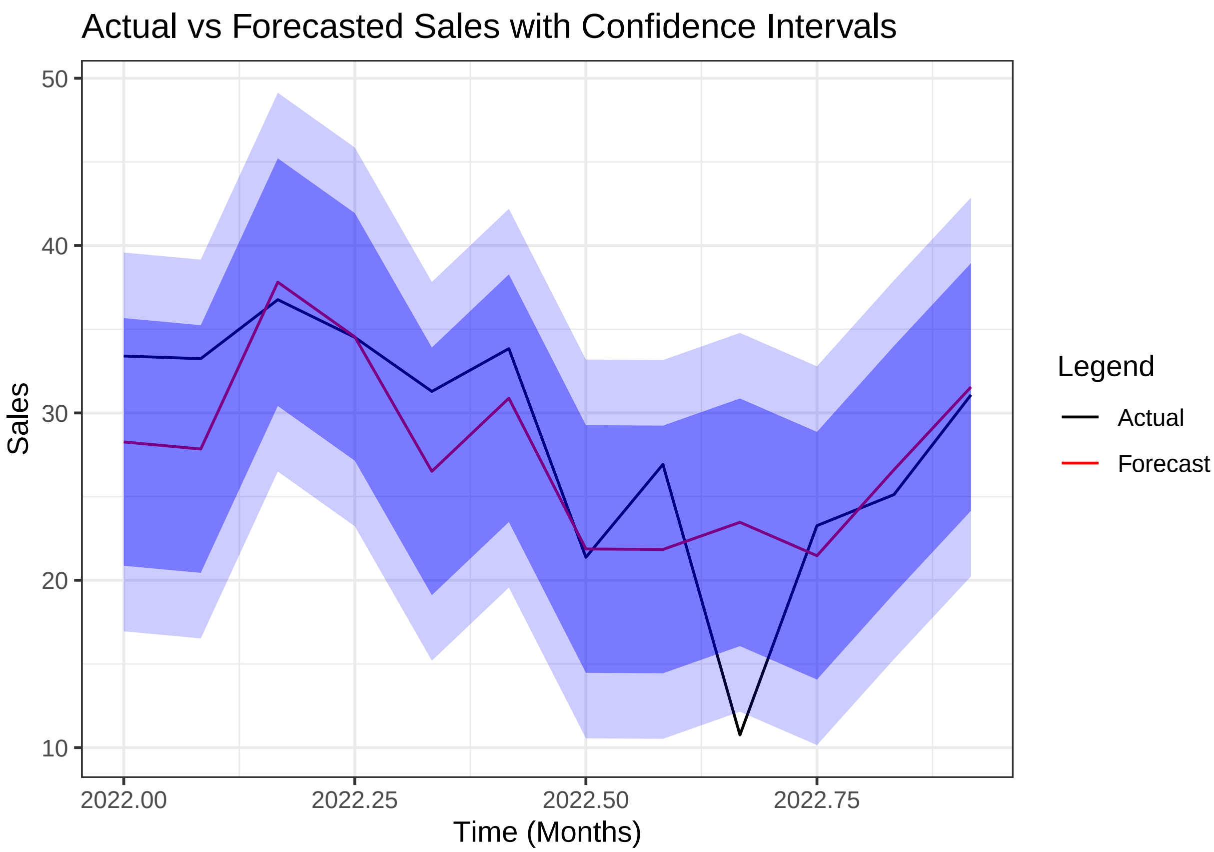
6. Plot Actual vs Forecasted Values with Confidence Intervals

Finally, I plot the actual test data against the forecasted values to visually assess the model's accuracy.

```
# Combine actual test data and forecasted data into a single dataframe
plot_data <- data.frame(
  Date = time(test_ts),                         # Dates from the test data
  Actual = as.numeric(test_ts),                 # Actual sales values
  Forecast = as.numeric(forecast_arima_test$mean),  # Forecasted values
  Lower_80 = as.numeric(forecast_arima_test$lower[, 1]),  # 80% CI lower bound
  Upper_80 = as.numeric(forecast_arima_test$upper[, 1]),  # 80% CI upper bound
  Lower_95 = as.numeric(forecast_arima_test$lower[, 2]),  # 95% CI lower bound
  Upper_95 = as.numeric(forecast_arima_test$upper[, 2])   # 95% CI upper bound
)

# Plot actual vs forecast with confidence intervals
ggplot(plot_data, aes(x = Date)) +
  geom_line(aes(y = Actual, color = "Actual")) +  # Plot actual data
```

```
  geom_line(aes(y = Forecast, color = "Forecast")) +  # Plot forecast data
  geom_ribbon(aes(ymin = Lower_95, ymax = Upper_95), fill = "blue", alpha = 0.2) +  # 95% CI
  geom_ribbon(aes(ymin = Lower_80, ymax = Upper_80), fill = "blue", alpha = 0.4) +  # 80% CI
  ggtitle("Actual vs Forecasted Sales with Confidence Intervals") +
  ylab("Sales") +
  xlab("Time (Months)") +
  scale_color_manual(name = "Legend", values = c("Actual" = "black", "Forecast" = "red"))+
  theme_bw()
```

```
## Don't know how to automatically pick scale for object of type <ts>. Defaulting
## to continuous.
```



Explanation:

- I create a dataframe that includes the actual values, forecasted values, and the confidence intervals.

- The plot displays the actual test data and the forecasted values, with shaded areas indicating the 80% and 95% confidence intervals, giving a clear visual representation of the model's performance.