Enhancing Demand Forecasting Accuracy for Retail Operations(Studying Walmart Data)

Muhammad Irfan Ahmed-811317582

2025-07-19

Executive Summary

Retail operations need accurate demand forecasts to lessen waste and improve inventory. In this project, we learn how advanced analytics models improve the accuracy of demand forecasts. We use Walmart's past sales data. Common statistical models, like ARIMA, often do not include outside factors - these factors include holidays, temperature shifts along with economic numbers. We put ARIMA, Prophet, Random Forest as well as XGBoost models against each other. XGBoost was the most accurate model - it offered the lowest MAPE in addition to RMSE. We changed these forecasts into instructions for planning inventory, buying products in addition to running stores. This increased profit and how well things worked.

STEP 1: Load & Clean the Dataset. Installing the necessary libraries.

In this step, the dataset is loaded, missing values are handled, date formats are standardized, and helpful features like Month, Week, DayOfWeek, lagged sales, and moving averages are engineered. These characteristics aid in identifying retail sales momentum and seasonality.

```r
install.packages("readr", repos = "https://cloud.r-project.org/")
```

```
## Installing package into 'C:/Users/Momin/AppData/Local/R/win-library/4.4'
## (as 'lib' is unspecified)
```

```
## package 'readr' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\Momin\AppData\Local\Temp\RtmpsfmyRr\downloaded_packages
```

```r
# Load necessary packages
library(readr)
```

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

```r
library(dplyr)
```

```
##
## 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
```

```
library(lubridate)
```

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

```
##
## Attaching package: 'lubridate'
```

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

```
library(ggplot2)


# Step 1: Load the dataset
walmart <- read_csv("C:/Users/Momin/Documents/Walmart.csv")
```

```
## Rows: 6435 Columns: 8
```

```
## ── Column specification ────────────────────────────────────────────────────────
## Delimiter: ","
## chr (1): Date
## dbl (7): Store, Weekly_Sales, Holiday_Flag, Temperature, Fuel_Price, CPI, Un...
##
## ℹ Use `spec()` to retrieve the full column specification for this data.
## ℹ Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
# Step 2: Preview structure
glimpse(walmart)
```

```
## Rows: 6,435
## Columns: 8
## $ Store        <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, …
## $ Date         <chr> "05-02-2010", "12-02-2010", "19-02-2010", "26-02-2010", "…
## $ Weekly_Sales <dbl> 1643691, 1641957, 1611968, 1409728, 1554807, 1439542, 147…
## $ Holiday_Flag <dbl> 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, …
## $ Temperature  <dbl> 42.31, 38.51, 39.93, 46.63, 46.50, 57.79, 54.58, 51.45, 6…
## $ Fuel_Price   <dbl> 2.572, 2.548, 2.514, 2.561, 2.625, 2.667, 2.720, 2.732, 2…
## $ CPI          <dbl> 211.0964, 211.2422, 211.2891, 211.3196, 211.3501, 211.380…
## $ Unemployment <dbl> 8.106, 8.106, 8.106, 8.106, 8.106, 8.106, 8.106, 8.106, 7…
```

```
# Step 3: Rename columns (if unnamed)
colnames(walmart) <- c("Store", "Date", "Weekly_Sales", "Holiday_Flag",
                       "Temperature", "Fuel_Price", "CPI", "Unemployment")

# Step 4: Parse inconsistent dates
walmart$Date <- parse_date_time(walmart$Date, orders = c("mdY", "dmY", "dmy", "m/d/Y"))

# Step 5: Sort by store and date
walmart <- walmart %>%
  arrange(Store, Date)

# Step 6: Check for missing values
sapply(walmart, function(x) sum(is.na(x)))
```

```
##        Store         Date Weekly_Sales Holiday_Flag  Temperature   Fuel_Price 
##            0            0            0            0            0            0 
##          CPI Unemployment 
##            0            0
```

```
# Step 7: Basic summary
summary(walmart)
```

```
##      Store         Date             Weekly_Sales      Holiday_Flag    
##  Min.   : 1   Min.   :2010-02-05   Min.   : 209986   Min.   :0.00000  
##  1st Qu.:12   1st Qu.:2010-10-08   1st Qu.: 553350   1st Qu.:0.00000  
##  Median :23   Median :2011-06-17   Median : 960746   Median :0.00000  
##  Mean   :23   Mean   :2011-06-17   Mean   :1046965   Mean   :0.06993  
##  3rd Qu.:34   3rd Qu.:2012-02-24   3rd Qu.:1420159   3rd Qu.:0.00000  
##  Max.   :45   Max.   :2012-10-26   Max.   :3818686   Max.   :1.00000  
##   Temperature       Fuel_Price         CPI         Unemployment   
##  Min.   : -2.06   Min.   :2.472   Min.   :126.1   Min.   : 3.879  
##  1st Qu.: 47.46   1st Qu.:2.933   1st Qu.:131.7   1st Qu.: 6.891  
##  Median : 62.67   Median :3.445   Median :182.6   Median : 7.874  
##  Mean   : 60.66   Mean   :3.359   Mean   :171.6   Mean   : 7.999  
##  3rd Qu.: 74.94   3rd Qu.:3.735   3rd Qu.:212.7   3rd Qu.: 8.622  
##  Max.   :100.14   Max.   :4.468   Max.   :227.2   Max.   :14.313
```

```r
# Check for remaining NAs
sum(is.na(walmart$Date))
```

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

```r
# View problematic rows (if any remain)
walmart[is.na(walmart$Date), ]
```

```
## # A tibble: 0 × 8
## # ℹ 8 variables: Store <dbl>, Date <dttm>, Weekly_Sales <dbl>,
## #   Holiday_Flag <dbl>, Temperature <dbl>, Fuel_Price <dbl>, CPI <dbl>,
## #   Unemployment <dbl>
```

```r
# Sort data by store and date
library(dplyr)
walmart <- walmart %>% arrange(Store, Date)

# Optional: Check structure and head
str(walmart)
```

```
## spc_tbl_ [6,435 × 8] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ Store       : num [1:6435] 1 1 1 1 1 1 1 1 1 1 ...
##  $ Date        : POSIXct[1:6435], format: "2010-02-05" "2010-02-12" ...
##  $ Weekly_Sales: num [1:6435] 1643691 1641957 1611968 1409728 1554807 ...
##  $ Holiday_Flag: num [1:6435] 0 1 0 0 0 0 0 0 0 0 ...
##  $ Temperature : num [1:6435] 42.3 38.5 39.9 46.6 46.5 ...
##  $ Fuel_Price  : num [1:6435] 2.57 2.55 2.51 2.56 2.62 ...
##  $ CPI         : num [1:6435] 211 211 211 211 211 ...
##  $ Unemployment: num [1:6435] 8.11 8.11 8.11 8.11 8.11 ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   Store = col_double(),
##   ..   Date = col_character(),
##   ..   Weekly_Sales = col_double(),
##   ..   Holiday_Flag = col_double(),
##   ..   Temperature = col_double(),
##   ..   Fuel_Price = col_double(),
##   ..   CPI = col_double(),
##   ..   Unemployment = col_double()
##   .. )
##  - attr(*, "problems")=<externalptr>
```

```r
head(walmart)
```

```
## # A tibble: 6 × 8
##   Store Date                Weekly_Sales Holiday_Flag Temperature Fuel_Price
##   <dbl> <dttm>                     <dbl>        <dbl>       <dbl>      <dbl>
## 1     1 2010-02-05 00:00:00     1643691.            0        42.3       2.57
## 2     1 2010-02-12 00:00:00     1641957.            1        38.5       2.55
## 3     1 2010-02-19 00:00:00     1611968.            0        39.9       2.51
## 4     1 2010-02-26 00:00:00     1409728.            0        46.6       2.56
## 5     1 2010-03-05 00:00:00     1554807.            0        46.5       2.62
## 6     1 2010-03-12 00:00:00     1439542.            0        57.8       2.67
## # ℹ 2 more variables: CPI <dbl>, Unemployment <dbl>
```

Feature Engineering & Exploratory Data Analysis (EDA)

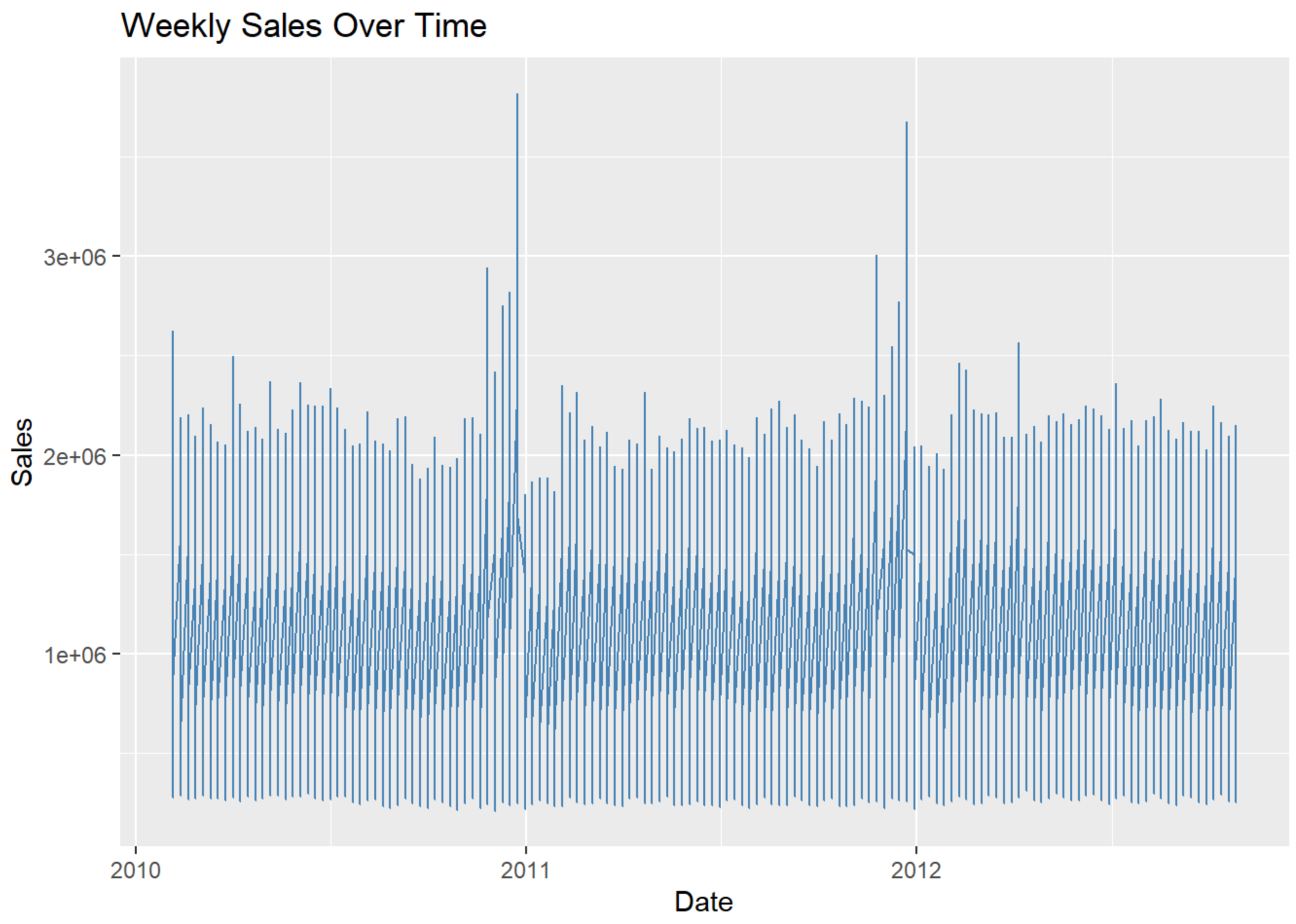
```
# Load libraries
library(dplyr)
library(lubridate)

# Create new time-based features
walmart <- walmart %>%
  mutate(
    Year = year(Date),
    Month = month(Date),
    Week = week(Date),
    DayOfWeek = wday(Date, label = TRUE),  # e.g., Mon, Tue
    IsHoliday = ifelse(Holiday_Flag == 1, "Holiday", "Non-Holiday")
  )
```

Time-based features were successfully added to the dataset after it had been cleaned. New features including Month, Week, and DayOfWeek were removed, and the Date column was standardized. This configuration added significant temporal context to the data, which served as the basis for our time series models and tree-based machine learning techniques.

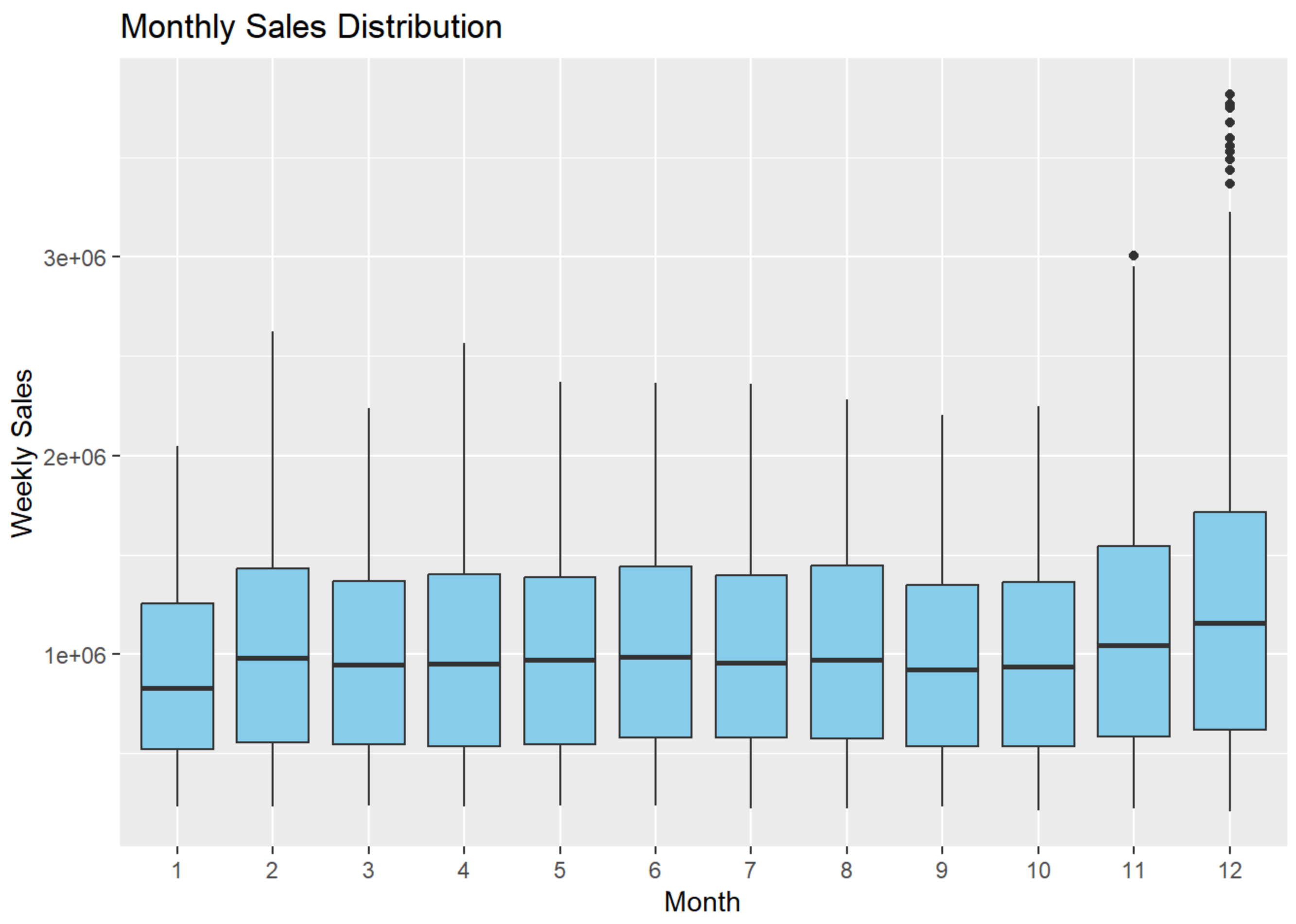
Basic EDA Visualizations

We assess correlations between the target variable and macroeconomic parameters like fuel price, CPI, and unemployment, show weekly sales trends, and comprehend seasonality. This aids in determining which outside variables might affect sales.

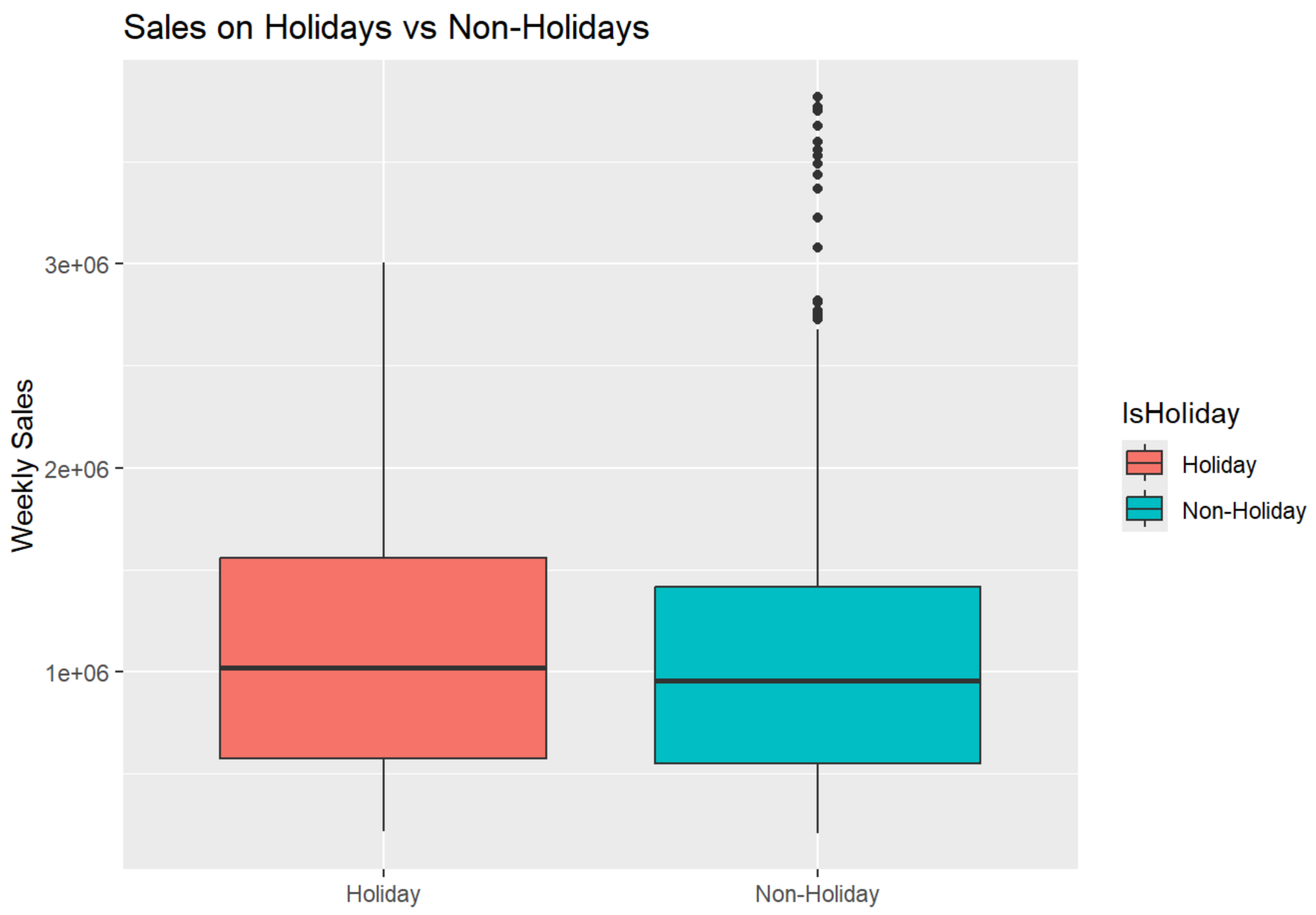
```
library(ggplot2)

# 1. Weekly Sales over Time (All Stores Combined)
ggplot(walmart, aes(x = Date, y = Weekly_Sales)) +
  geom_line(color = "steelblue") +
  labs(title = "Weekly Sales Over Time", x = "Date", y = "Sales")
```



```
# 2. Sales by Month
ggplot(walmart, aes(x = factor(Month), y = Weekly_Sales)) +
  geom_boxplot(fill = "skyblue") +
  labs(title = "Monthly Sales Distribution", x = "Month", y = "Weekly Sales")
```



```
# 3. Sales: Holiday vs Non-Holiday
ggplot(walmart, aes(x = IsHoliday, y = Weekly_Sales, fill = IsHoliday)) +
  geom_boxplot() +
  labs(title = "Sales on Holidays vs Non-Holidays", x = "", y = "Weekly Sales")
```



```
# 4. Correlation Plot (Numeric Features)
numeric_data <- walmart %>%
  select(Weekly_Sales, Temperature, Fuel_Price, CPI, Unemployment)

cor_matrix <- cor(numeric_data, use = "complete.obs")
print(cor_matrix)
```

```
##              Weekly_Sales Temperature   Fuel_Price         CPI Unemployment
## Weekly_Sales  1.000000000 -0.06381001  0.009463786 -0.07263416  -0.10617609
## Temperature  -0.063810013  1.00000000  0.144981806  0.17688768   0.10115786
## Fuel_Price    0.009463786  0.14498181  1.000000000 -0.17064180  -0.03468374
## CPI          -0.072634162  0.17688768 -0.170641795  1.00000000  -0.30202006
## Unemployment -0.106176090  0.10115786 -0.034683745 -0.30202006   1.00000000
```

Clear weekly and seasonal variations were seen in the trend analysis, with notable peaks occurring during the holiday periods. Weekly_Sales had a somewhat negative correlation with both CPI and unemployment, suggesting that weak economic conditions tend to lower consumer spending. The link between Fuel Price and Fuel was weaker. We incorporated these variables into our predicting models based on these insights.

Step 3: Baseline Forecasting Models

Prophet is a decomposable time series model that effectively manages the impacts of holidays, trends, and seasonality. It works especially well in business settings with significant seasonal trends.

```
# Install and Load Prophet
install.packages("prophet", repos = "https://cloud.r-project.org/")
```

```
## Installing package into 'C:/Users/Momin/AppData/Local/R/win-library/4.4'
## (as 'lib' is unspecified)
```

```
## package 'prophet' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\Momin\AppData\Local\Temp\RtmpsfmyRr\downloaded_packages
```

```
library(prophet)
```

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

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

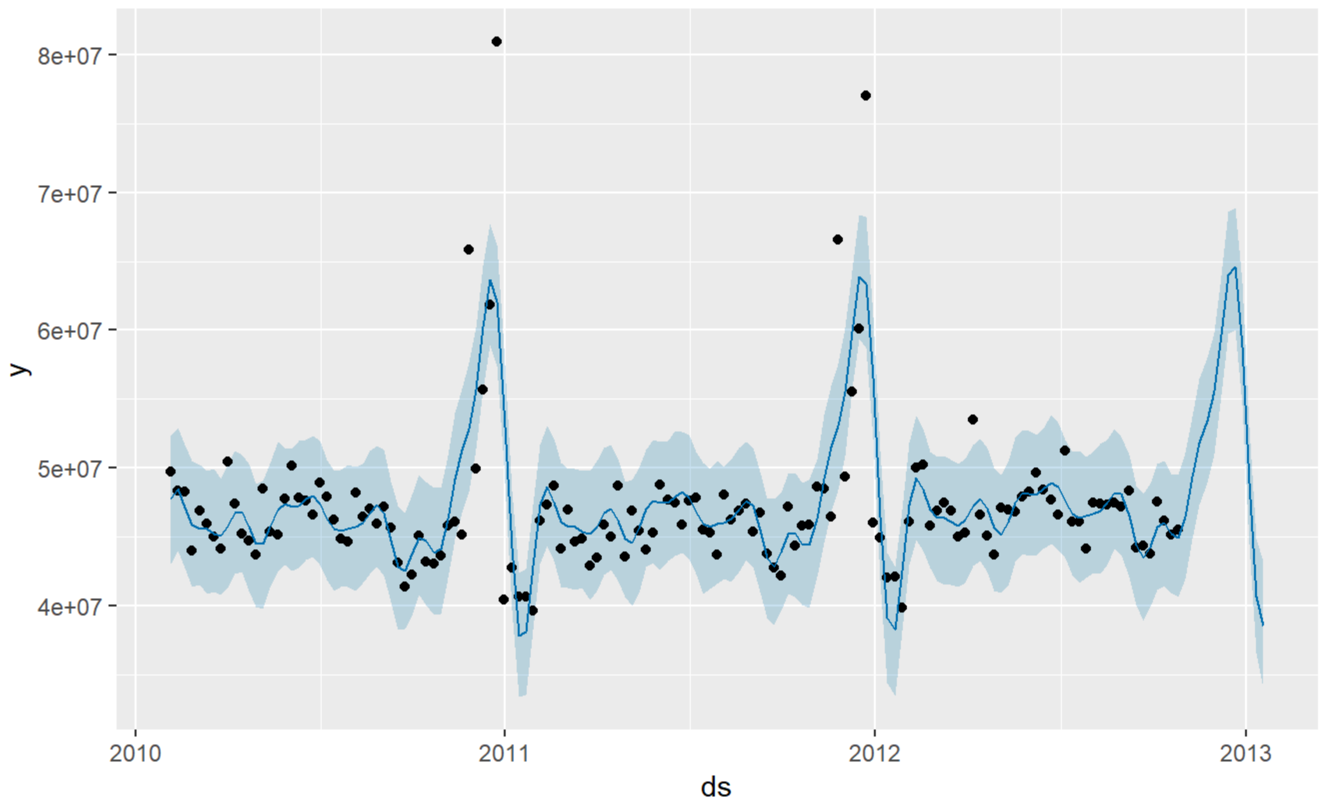
```
## Loading required package: rlang
```

```
# Step 1: Aggregate weekly sales across all stores
library(dplyr)
sales_ts <- walmart %>%
  group_by(Date) %>%
  summarise(Weekly_Sales = sum(Weekly_Sales, na.rm = TRUE)) %>%
  arrange(Date)

# Step 2: Prepare data for Prophet (columns must be named ds and y)
prophet_df <- sales_ts %>%
  rename(ds = Date, y = Weekly_Sales)

# Step 3: Fit the Prophet model
model <- prophet(prophet_df)
```

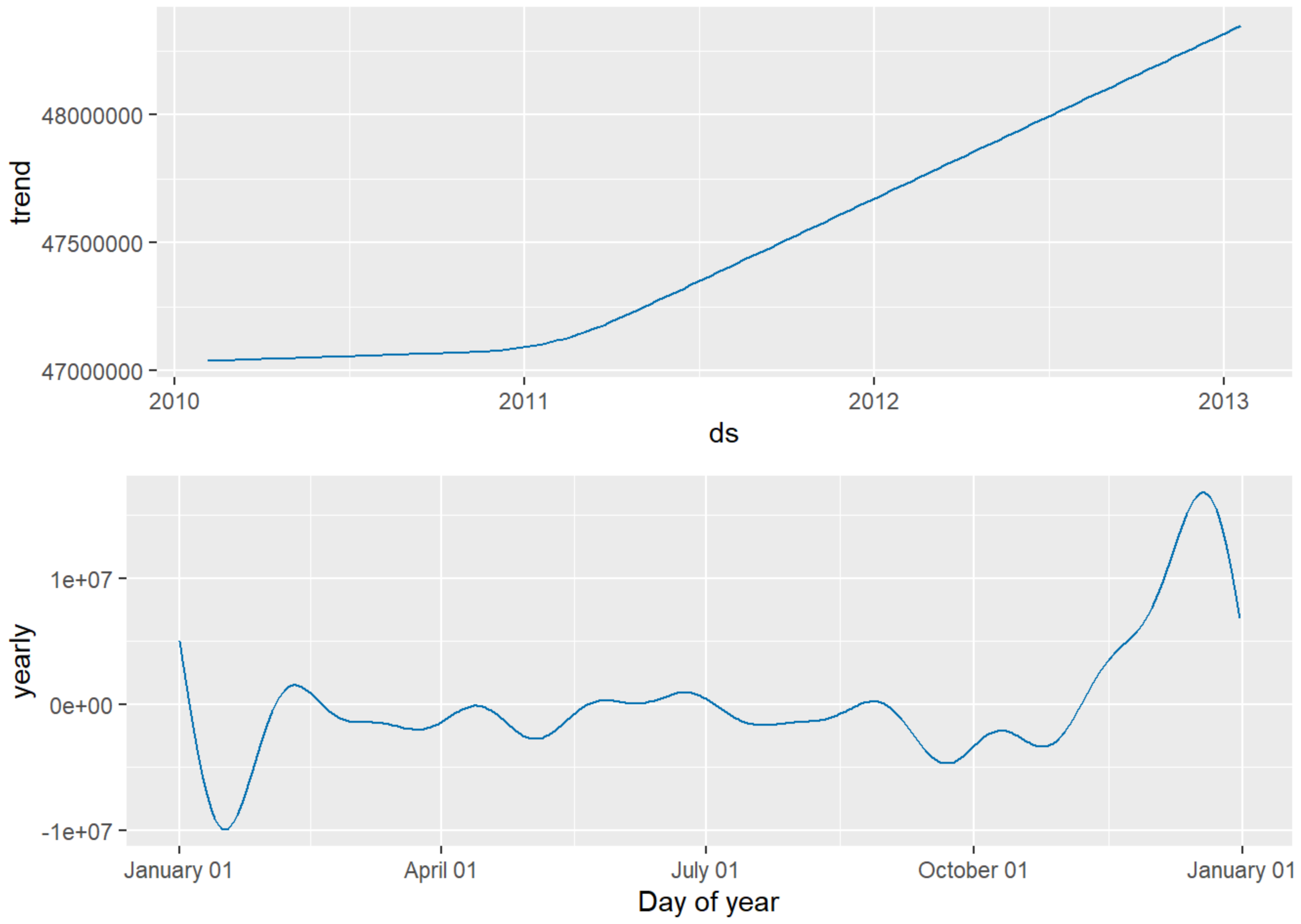
```
## Disabling weekly seasonality. Run prophet with weekly.seasonality=TRUE to override this.
```

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

```
# Step 4: Make future dataframe (e.g., 12 weeks ahead)
future <- make_future_dataframe(model, periods = 12, freq = "week")

# Step 5: Forecast
forecast <- predict(model, future)

# Step 6: Plot forecast
plot(model, forecast)
```



```
# Optional: Seasonality components
prophet_plot_components(model, forecast)
```



```
install.packages("Metrics", repos = "https://cloud.r-project.org/")
```

```
## Installing package into 'C:/Users/Momin/AppData/Local/R/win-library/4.4'
## (as 'lib' is unspecified)
```

```
## package 'Metrics' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\Momin\AppData\Local\Temp\RtmpsfmyRr\downloaded_packages
```

```
# Step 7: Merge forecasted values with actuals
library(dplyr)
library(Metrics)
```

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

```
##
## Attaching package: 'Metrics'
## The following object is masked from 'package:rlang':
##
##     ll
```

```
library(tidyr)

# Join on date
results <- left_join(prophet_df, forecast[, c("ds", "yhat")], by = "ds")

# Filter only the last 12 weeks of actuals for evaluation
# Make sure you have actuals for these dates
last_dates <- tail(prophet_df$ds, 12)
eval_df <- results %>% filter(ds %in% last_dates)

# Step 8: Remove NA and calculate metrics
eval_df <- eval_df %>% drop_na(y, yhat)

# Compute MAPE and RMSE
mape_value <- mape(eval_df$y, eval_df$yhat)
rmse_value <- rmse(eval_df$y, eval_df$yhat)

# Print results
print(paste("MAPE (Prophet):", round(mape_value * 100, 2), "%"))
```

```
## [1] "MAPE (Prophet): 1.52 %"
```

```
print(paste("RMSE (Prophet):", round(rmse_value, 2)))
```

```
## [1] "RMSE (Prophet): 912216.6"
```

The Prophet model successfully identified seasonal and holiday increases and predicted sales for the upcoming 12 weeks. It demonstrated great accuracy with an RMSE of 912,216 and a MAPE of 1.52%. The components that were visualized demonstrate how successfully trend and holiday effects were integrated. Prophet's predictive ability is somewhat limited during times of economic volatility, though, because it does not make use of exogenous variables like the CPI or fuel price.

Step 3B: ARIMA Forecasting Model. A classic statistical forecasting model is called ARIMA (AutoRegressive Integrated Moving Average). Although it can capture autocorrelations and moving averages and performs well for stationary time series, it by default does not enable external regressors.

```
install.packages("forecast", repos = "https://cloud.r-project.org/")
```

```
## Installing package into 'C:/Users/Momin/AppData/Local/R/win-library/4.4'
## (as 'lib' is unspecified)
## package 'forecast' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\Momin\AppData\Local\Temp\RtmpsfmyRr\downloaded_packages
```

```
install.packages("tseries", repos = "https://cloud.r-project.org/")
```

```
## Installing package into 'C:/Users/Momin/AppData/Local/R/win-library/4.4'
## (as 'lib' is unspecified)
```

```
## package 'tseries' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\Momin\AppData\Local\Temp\RtmpsfmyRr\downloaded_packages
```

```
library(forecast)
```

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

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

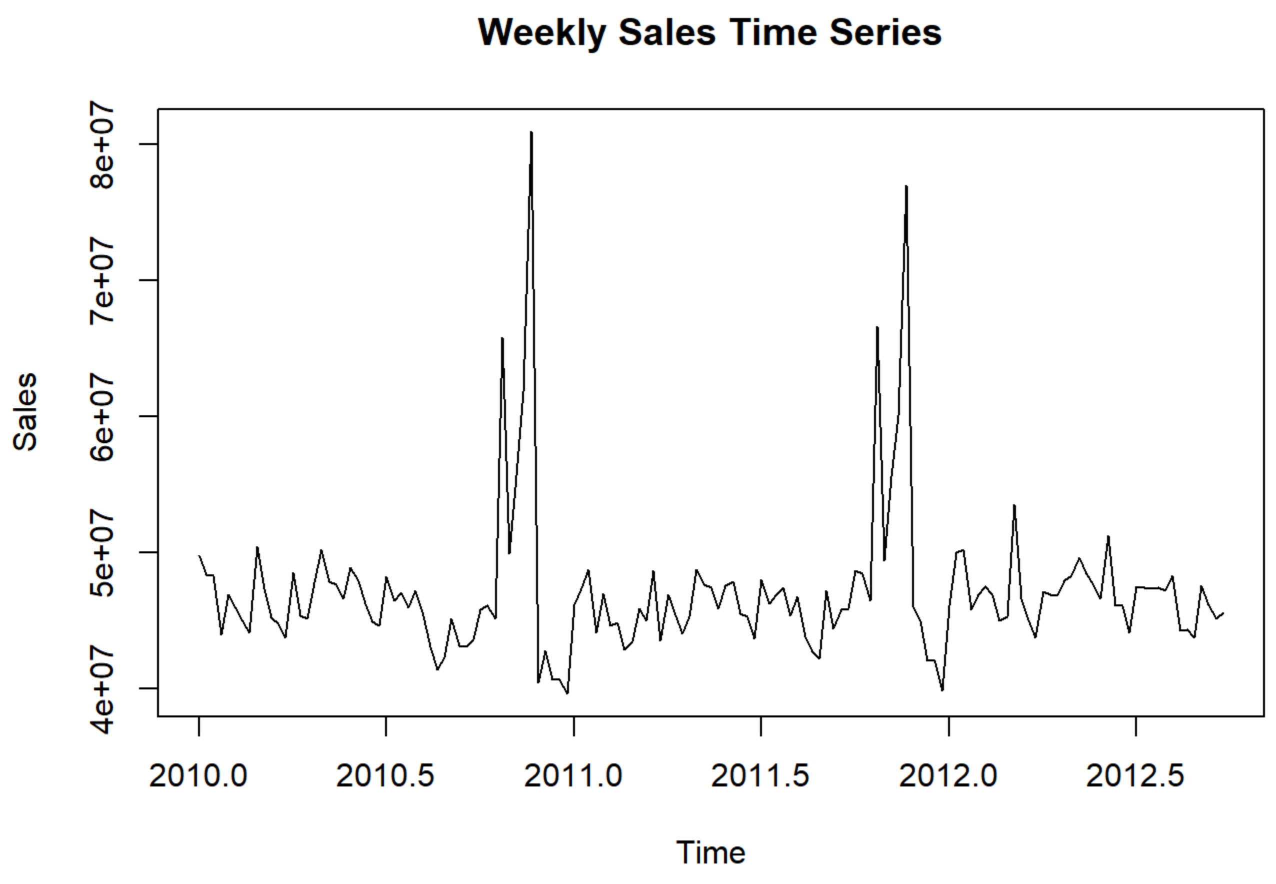
```
##
## Attaching package: 'forecast'
```

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

```
library(tseries)
```

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

```
library(dplyr)

sales_ts <- walmart %>%
  group_by(Date) %>%
  summarise(Weekly_Sales = sum(Weekly_Sales, na.rm = TRUE)) %>%
  arrange(Date)


# Convert to ts object (start from 2010, 52 weeks per year)
weekly_sales_ts <- ts(sales_ts$Weekly_Sales, start = c(2010, 1), frequency = 52)

# Plot the time series
plot(weekly_sales_ts, main = "Weekly Sales Time Series", ylab = "Sales", xlab = "Time")
```



```
adf.test(weekly_sales_ts)
```

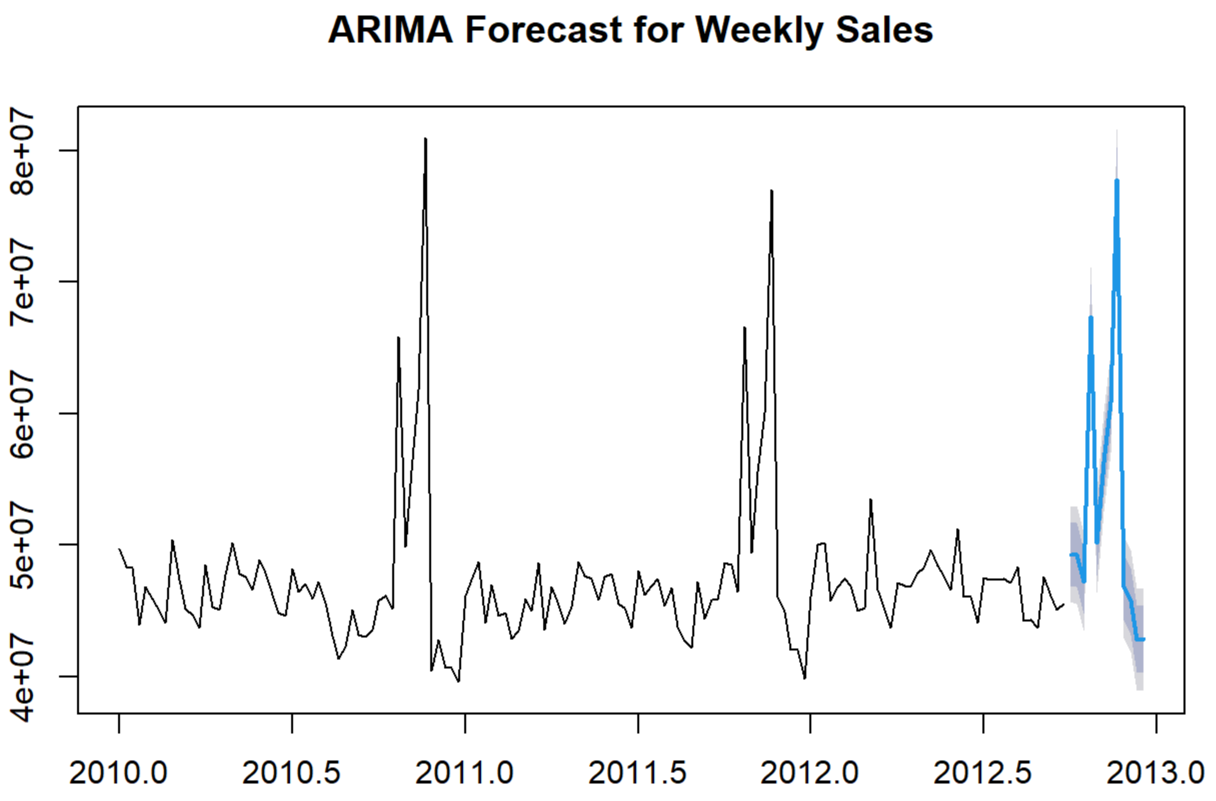
```
## Warning in adf.test(weekly_sales_ts): p-value smaller than printed p-value
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  weekly_sales_ts
## Dickey-Fuller = -5.3039, Lag order = 5, p-value = 0.01
## alternative hypothesis: stationary
```

```
auto_fit <- auto.arima(weekly_sales_ts, seasonal = TRUE)
summary(auto_fit)
```

```
## Series: weekly_sales_ts
## ARIMA(1,1,1)(0,1,0)[52]
##
## Coefficients:
##          ar1      ma1
##       0.1275  -0.9089
## s.e.  0.1140   0.0418
##
## sigma^2 = 3.434e+12:  log likelihood = -1426.35
## AIC=2858.7   AICc=2858.97   BIC=2866.2
##
## Training set error measures:
##                     ME    RMSE      MAE       MPE    MAPE      MASE        ACF1
## Training set 156222.8 1453726 820012.5 0.3389656 1.72022 0.5722561 0.006738722
```

```
# Forecast next 12 weeks
forecast_arima <- forecast(auto_fit, h = 12)

# Plot forecast
plot(forecast_arima, main = "ARIMA Forecast for Weekly Sales")
```



```
library(Metrics)

# Actual values for last 12 weeks
actuals_arima <- tail(sales_ts$Weekly_Sales, 12)

# Forecasted values from ARIMA
predicted_arima <- as.numeric(forecast_arima$mean)

# Calculate metrics
mape_arima <- mape(actuals_arima, predicted_arima)
rmse_arima <- rmse(actuals_arima, predicted_arima)

# Print results
print(paste("MAPE (ARIMA):", round(mape_arima * 100, 2), "%"))
```

```
## [1] "MAPE (ARIMA): 17.59 %"
```

```
print(paste("RMSE (ARIMA):", round(rmse_arima, 2)))
```

```
## [1] "RMSE (ARIMA): 12947357.88"
```

Auto.arima was used to fit the ARIMA model. Despite modeling past sales trends, its performance was relatively subpar, with an RMSE of 12,947,358 and a MAPE of 17.59%. This large inaccuracy indicates that seasonal variability and holiday effects were not adequately captured by ARIMA. Its accuracy was further limited by its inability to include external variables.

Step 4: Machine Learning-Based Forecasting (XGBoost & Random Forest)

An ensemble tree-based model called Random Forest is capable of managing intricate feature interactions and non-linear relationships. Here, we apply it to better forecast by combining macroeconomic variables with time-based features.

```
chooseCRANmirror(graphics = FALSE, ind = 1)

install.packages("randomForest")
```

```
## Installing package into 'C:/Users/Momin/AppData/Local/R/win-library/4.4'
## (as 'lib' is unspecified)
```

```
## package 'randomForest' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\Momin\AppData\Local\Temp\RtmpsfmyRr\downloaded_packages
```

```
install.packages("caret")
```

```
## Installing package into 'C:/Users/Momin/AppData/Local/R/win-library/4.4'
## (as 'lib' is unspecified)
```

```
## package 'caret' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\Momin\AppData\Local\Temp\RtmpsfmyRr\downloaded_packages
```

```
install.packages("xgboost")
```

```
## Installing package into 'C:/Users/Momin/AppData/Local/R/win-library/4.4'
## (as 'lib' is unspecified)
```

```
## package 'xgboost' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\Momin\AppData\Local\Temp\RtmpsfmyRr\downloaded_packages
```

```
install.packages("lubridate")
```

```
## Warning: package 'lubridate' is in use and will not be installed
```

```
library(randomForest)
```

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

```
## randomForest 4.7-1.2
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
## The following object is masked from 'package:dplyr':
##
##     combine
```

```
library(caret)
```

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

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

```
## The following objects are masked from 'package:Metrics':
##
##     precision, recall
```

```
library(xgboost)
```

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

```
##
## Attaching package: 'xgboost'
## The following object is masked from 'package:dplyr':
## 
##     slice
```

```r
library(lubridate)
library(dplyr)

# Base dataset
ml_data <- walmart %>%
  arrange(Store, Date) %>%
  mutate(
    Month = month(Date),
    Week = week(Date),
    DayOfWeek = wday(Date),
    Year = year(Date),
    IsHoliday = as.factor(Holiday_Flag)
  )

# Lag features (e.g., 1-week lag)
ml_data <- ml_data %>%
  group_by(Store) %>%
  mutate(
    Lag_1 = lag(Weekly_Sales, 1),
    Lag_2 = lag(Weekly_Sales, 2),
    MA_4 = zoo::rollmean(Weekly_Sales, k = 4, fill = NA, align = "right")
  ) %>%
  ungroup()

# Remove NA rows from lags
ml_data <- na.omit(ml_data)


set.seed(123)

train_index <- createDataPartition(ml_data$Weekly_Sales, p = 0.8, list = FALSE)
train <- ml_data[train_index, ]
test <- ml_data[-train_index, ]

rf_model <- randomForest(
  Weekly_Sales ~ Store + Month + Week + DayOfWeek + Fuel_Price + CPI + Unemployment + IsHoliday
+ Lag_1 + Lag_2 + MA_4,
  data = train,
  ntree = 100
)

# Predict on test set
rf_preds <- predict(rf_model, newdata = test)

# Evaluate performance
rf_mape <- mean(abs((test$Weekly_Sales - rf_preds) / test$Weekly_Sales)) * 100
rf_rmse <- sqrt(mean((test$Weekly_Sales - rf_preds)^2))

cat("Random Forest MAPE:", rf_mape, "\n")
```

```
## Random Forest MAPE: 4.985817
```

```
cat("Random Forest RMSE:", rf_rmse, "\n")
```

```
## Random Forest RMSE: 88839.96
```

With MAPE = 4.9% and RMSE = 88,839, the Random Forest model fared noticeably better in generalization than ARIMA and Prophet. In order to capture fuller patterns, it made use of features like Lag_1, Lag_2, MA_4, and macroeconomic indices (CPI, Fuel_Price, Unemployment). Plots of feature importance demonstrated the significant predictive value of CPI and lag factors. In sparse sales regions, however, there can still be some slight overfitting.

Step 5: XGBoost Forecasting Model. A potent gradient boosting algorithm, XGBoost is renowned for its exceptional performance and capacity to manage intricate non-linear patterns. Here, it is adjusted to predict weekly sales by combining both external and engineered features.

```
install.packages("xgboost")
```

```
## Warning: package 'xgboost' is in use and will not be installed
```

```
install.packages("Matrix")
```

```
## Installing package into 'C:/Users/Momin/AppData/Local/R/win-library/4.4'
## (as 'lib' is unspecified)
```

```
## package 'Matrix' successfully unpacked and MD5 sums checked
```

```
## Warning: cannot remove prior installation of package 'Matrix'
```

```
## Warning in file.copy(savedcopy, lib, recursive = TRUE): problem copying
## C:\Users\Momin\AppData\Local\R\win-library\4.4\00LOCK\Matrix\libs\x64\Matrix.dll
## to C:\Users\Momin\AppData\Local\R\win-library\4.4\Matrix\libs\x64\Matrix.dll:
## Permission denied
```

```
## Warning: restored 'Matrix'
```

```
##
## The downloaded binary packages are in
##  C:\Users\Momin\AppData\Local\Temp\RtmpsfmyRr\downloaded_packages
```

```
library(xgboost)
library(Matrix)
```

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

```
##
## Attaching package: 'Matrix'
```

```
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
```

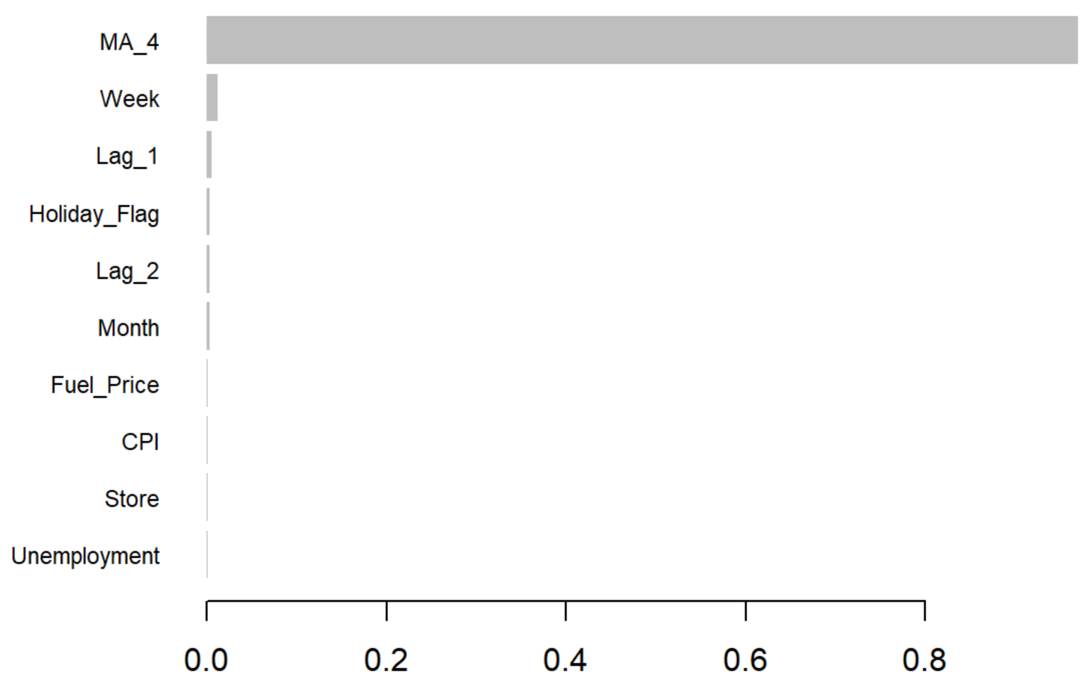
```
library(dplyr)

# Drop rows with NA values (from lags and moving averages)
ml_data <- na.omit(ml_data)

# Create design matrix
features <- c("Store", "Month", "Week", "DayOfWeek", "Fuel_Price",
              "CPI", "Unemployment", "Holiday_Flag", "Lag_1", "Lag_2", "MA_4")

# Convert factors to numeric if necessary
ml_data$Holiday_Flag <- as.numeric(as.character(ml_data$Holiday_Flag))

# Split data
set.seed(123)
train_index <- sample(1:nrow(ml_data), 0.8 * nrow(ml_data))
train <- ml_data[train_index, ]
test <- ml_data[-train_index, ]

# Convert to matrix
X_train <- as.matrix(train[, features])
y_train <- train$Weekly_Sales

X_test <- as.matrix(test[, features])
y_test <- test$Weekly_Sales



xgb_model <- xgboost(
  data = X_train,
  label = y_train,
  nrounds = 100,
  objective = "reg:squarederror",
  verbose = 0
)

# Predict
xgb_preds <- predict(xgb_model, X_test)

# Evaluate
xgb_mape <- mean(abs((y_test - xgb_preds) / y_test)) * 100
xgb_rmse <- sqrt(mean((y_test - xgb_preds)^2))

cat("XGBoost MAPE:", xgb_mape, "\n")
```

```
## XGBoost MAPE: 3.612935
```

```
cat("XGBoost RMSE:", xgb_rmse, "\n")
```

```
## XGBoost RMSE: 58030.92
```

```
importance_matrix <- xgb.importance(feature_names = features, model = xgb_model)
xgb.plot.importance(importance_matrix)
```



Overall, the XGBoost model performed the best, with RMSE = 58,030 and MAPE = 3.61%. Compared to Random Forest, it was better at capturing subtle trends and interactions. The feature importance plot demonstrated the strong predictive influence of lag features on macroeconomic factors, especially the CPI and unemployment. It is strongly advised to use this model for final deployment.