Logistics and Supply Chain Analytics

PROBLEM STATEMENT

The dataset provided by one of the largest fast-food restaurant chains in the US encompasses various aspects crucial for operational insights, including transaction details, menu item compositions, and restaurant metadata. Spanning from early March 2015 to June 15, 2015, the dataset encompasses transactional records from four distinct stores located in Berkeley, CA, and New York, NY. The task at hand is to anticipate the daily demand for the subsequent two weeks, extending from June 16, 2015, to June 29, 2015.

The primary objective is to aid restaurant managers in making informed inventory replenishment decisions, with a specific focus on forecasting the demand for lettuce, a vital ingredient, across the four individual restaurant locations. By providing accurate predictions for lettuce demand, tailored to each restaurant, this analysis aims to optimize inventory management and enhance operational efficiency within the fast-food chain.

DATA ANALYSIS AND PREPROCESSING

The data processing and analysis stage involved comprehensive exploration and integration of eleven distinct tables, each contributing unique insights into the operational dynamics of the fast-food restaurant chain. By identifying and illustrating the intricate relationships among the tables of the dataset, an ER diagram was crafted. The ER diagram provided a comprehensive overview of the interconnections between tables, laying the groundwork for subsequent data integration and preprocessing developments.

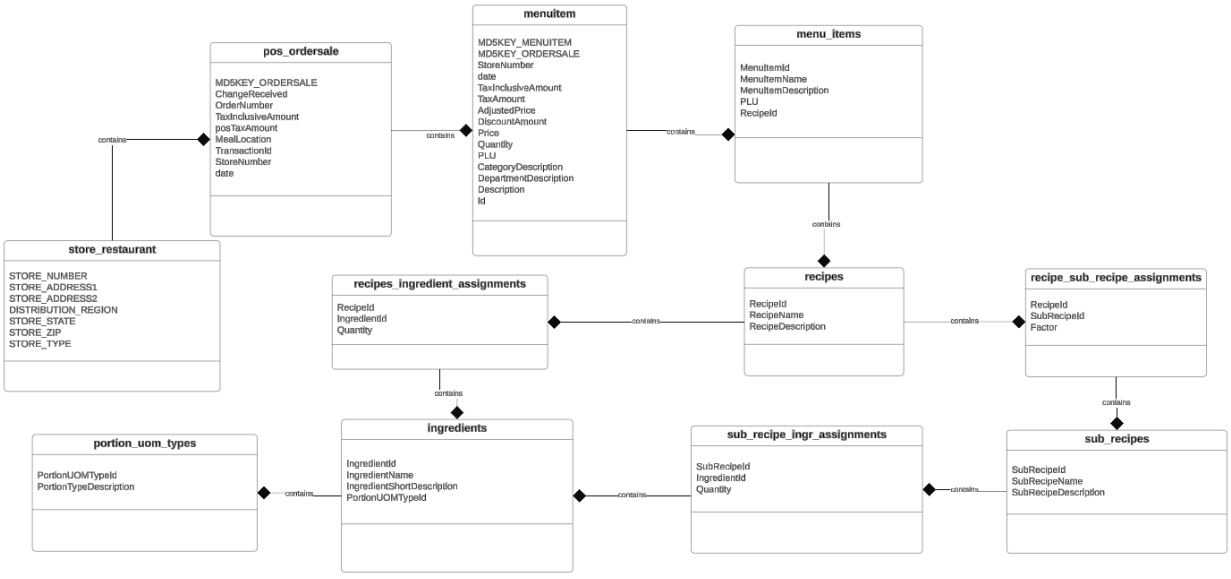


Fig 1. ER Diagram

FORECASTING MODELS

In undertaking the task of forecasting daily demand for the forthcoming two weeks (from 06/16/2015 to 06/29/2015) to inform inventory replenishment decisions, we employ two robust time series forecasting methodologies: Holt-Winters and ARIMA. These predictive models serve as invaluable tools for anticipating future lettuce demand, catering to the unique dynamics of each of the four individual restaurants under consideration.

CODE ANALYSIS OF THE RMD FILE

Code: Install all necessary packages

```
library(readr)
```

```
## Warning: package 'readr' was built under R version 4.3.2
```

```
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(forecast)
```

```
## Warning: package 'forecast' was built under R version 4.3.3

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

```
library(tseries)
```

```
## Warning: package 'tseries' was built under R version 4.3.3
```

```
library(ggplot2)
```

Load the provided Datasets

```
#Load datasets:

# Set the working directory to the location of your CSV files
setwd("C:/Users/Nikita Bhilare/OneDrive/Desktop/Imperial/Semester2/Logistics & Supply Chain/data/data")

# Load each CSV file into a separate data frame
pos_ordersale <- read_csv("pos_ordersale.csv")
```

```
## Rows: 43228 Columns: 9
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (2): MD5KEY_ORDERSALE, date
## dbl (7): ChangeReceived, OrderNumber, TaxInclusiveAmount, TaxAmount, MealLoc...
##
## 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.
```

```r
menuitem <- read_csv("menuitem.csv")
```

```
## Rows: 91431 Columns: 15
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (6): MD5KEY_MENUITEM, MD5KEY_ORDERSALE, CategoryDescription, DepartmentD...
## dbl (9): StoreNumber, TaxInclusiveAmount, TaxAmount, AdjustedPrice, Discount...
##
## 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.
```

```r
store_restaurant <- read_csv("store_restaurant.csv")
```

```
## Rows: 4 Columns: 9
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (7): STORE_ADDRESS1, STORE_ADDRESS2, DISTRIBUTION_REGION, STORE_STATE, S...
## dbl (2): STORE_ZIP, STORE_NUMBER
##
## 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.
```

```r
menu_items <- read_csv("menu_items.csv")
```

```
## Rows: 8005 Columns: 5
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (2): MenuItemName, MenuItemDescription
## dbl (3): PLU, MenuItemId, RecipeId
##
## 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.
```

```r
recipes <- read_csv("recipes.csv")
```

```
## Rows: 7458 Columns: 3
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (2): RecipeName, RecipeDescription
## dbl (1): RecipeId
##
## 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.
```

```r
recipes_ingredient_assignments <- read_csv("recipe_ingredient_assignments.csv")
```

```
## Rows: 14308 Columns: 3
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (3): RecipeId, IngredientId, Quantity
##
## 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.
```

```r
recipe_sub_recipe_assignments <- read_csv("recipe_sub_recipe_assignments.csv")
```

```
## Rows: 13890 Columns: 3
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (3): RecipeId, SubRecipeId, Factor
##
## 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.
```

```r
sub_recipes <- read_csv("sub_recipes.csv")
```

```
## Rows: 81 Columns: 3
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (2): SubRecipeName, SubRecipeDescription
## dbl (1): SubRecipeId
##
## 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.
```

```r
sub_recipe_ingr_assignments <- read_csv("sub_recipe_ingr_assignments.csv")
```

```
## Rows: 379 Columns: 3
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (3): SubRecipeId, IngredientId, Quantity
##
## 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.
```

```r
ingredients <- read_csv("ingredients.csv")
```

```
## Rows: 673 Columns: 4
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (2): IngredientName, IngredientShortDescription
## dbl (2): IngredientId, PortionUOMTypeId
##
## 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.
```

```r
portion_uom_types <- read_csv("portion_uom_types.csv")
```

```
## Rows: 5 Columns: 2
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): PortionTypeDescription
## dbl (1): PortionUOMTypeId
##
## 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.
```

Ingredient Identifications: Focusing on retrieving rows from Recipes and SubRecipes for the ingrediants - 'Lettuce' and 'Lettuce - Metric', as we aim to deal with predicting the lettuce quantities for the entire process of our forecasting. The ingredient ids for the following are: 27 and 291 respectively. The newly created dataframe is then integrated with the recipes table and the SubRecipes tables seperately two create two new seperate dataframes.

```
# Step 1: Filter ingredients for lettuce
lettuce_df <- ingredients %>%
  filter(IngredientId %in% c(27, 291))

# Step 2: Join with recipes_ingredient_assignments
recipe_data <- inner_join(lettuce_df, recipes_ingredient_assignments, by = "IngredientId")

# Step 3: Join with recipies
recipe_data <- inner_join(recipe_data, recipes, by = "RecipeId")


# Step 4: Join with sub_recipe_ingr_assignments
subrecipe_data <- inner_join(lettuce_df, sub_recipe_ingr_assignments, by = "IngredientId")

# Step 5: Join with sub_recipes
subrecipe_data <- inner_join(subrecipe_data, sub_recipes, by = "SubRecipeId")

subrecipe_data <- subrecipe_data %>%
  rename(SubRecipe_ingr_quantity = Quantity)

# Step 6: Join recipes_data with subrecipes_data via recipe_sub_recipe_assignments
new_df <- inner_join(subrecipe_data, recipe_sub_recipe_assignments, by = "SubRecipeId")

#Step 7: Combine all rows for recipes and subrecipes dataframes
lettuce_usage <- bind_rows(recipe_data, new_df)
```

Integration: Here we integrate all tables which include the sales, stores, and menu information. Furthermore, we integrate it with our lettuce_usage table that we created.

```
store_restaurant <- store_restaurant %>%
  rename(StoreNumber = STORE_NUMBER)

sales_df <- inner_join(store_restaurant, pos_ordersale, by = "StoreNumber")
sales_df <- inner_join(sales_df, menuitem, by = c("MD5KEY_ORDERSALE", "date", "StoreNumber"))

sales_df <- sales_df %>%
  rename(purchase_quantity = Quantity)

sales_df <- sales_df %>%
  rename(MenuItemId = Id)

sales_df <- inner_join(sales_df, menu_items, by = c("MenuItemId", "PLU"))

final <- inner_join(sales_df, lettuce_usage, by = "RecipeId")
```

```
## Warning in inner_join(sales_df, lettuce_usage, by = "RecipeId"): Detected an unexpected many-to-many
## i Row 262 of `x` matches multiple rows in `y`.
```

```
## i Row 396 of ‘y‘ matches multiple rows in ‘x‘.
## i If a many-to-many relationship is expected, set ‘relationship =
##   "many-to-many"‘ to silence this warning.
```

Column Selection: Almost all columns are present in the dataframe. Not all of them will be needed in the analysis later. Hence, to optimize our processes, Choosing columns which are necessary and storing them in our final dataset called predict_df is essential.

```
desired_column_order <- c(
  "MD5KEY_ORDERSALE", "date", "StoreNumber", "STORE_STATE", "MenuItemId", "purchase_quantity",
  "IngredientId", "IngredientName", "RecipeId", "Quantity",
  "SubRecipeId", "SubRecipe_ingr_quantity", "Factor"
)

# Rearrange the columns in the joined_data dataframe
predict_df <- final %>%
  select(desired_column_order)
```

```
## Warning: Using an external vector in selections was deprecated in tidyselect 1.1.0.
## i Please use ‘all_of()‘ or ‘any_of()‘ instead.
##   # Was:
##   data %>% select(desired_column_order)
##
##   # Now:
##   data %>% select(all_of(desired_column_order))
##
## See <https://tidyselect.r-lib.org/reference/faq-external-vector.html>.
## This warning is displayed once every 8 hours.
## Call ‘lifecycle::last_lifecycle_warnings()‘ to see where this warning was
## generated.
```

```
# Assuming 'predict_df' is your data frame
predict_df <- predict_df %>%
  mutate(
    recipe_ingr_quantity = replace(Quantity, is.na(Quantity), 0),
    SubRecipe_ingr_quantity = replace(SubRecipe_ingr_quantity, is.na(SubRecipe_ingr_quantity), 0), Facto
  )
```

Lettuce Calculation per store per day: we are calculating the lettuce used for each date, for each store.

```
# Assuming 'predict_df' is your data frame
total_lettuce_per_recipe <- predict_df %>%
  group_by(date, StoreNumber, MenuItemId, MD5KEY_ORDERSALE) %>%
  summarise(
    Quantity = sum(purchase_quantity *
    ((SubRecipe_ingr_quantity * Factor)
     + recipe_ingr_quantity),
    na.rm = TRUE)
  )
```

```
## ‘summarise()‘ has grouped output by ’date’, ’StoreNumber’, ’MenuItemId’. You
## can override using the ‘.groups‘ argument.
```

```r
# Assuming 'total_lettuce_per_recipe' is your data frame
total_lettuce_per_transaction <- total_lettuce_per_recipe %>%
  group_by(date, StoreNumber) %>%
  summarise(
    Total_Lettuce = sum(Quantity, na.rm = TRUE)
  ) %>%
  arrange(StoreNumber, date)
```

```
## `summarise()` has grouped output by 'date'. You can override using the
## `.groups` argument.
```

```r
head(total_lettuce_per_transaction)
```

```
## # A tibble: 6 x 3
## # Groups:   date [6]
##   date     StoreNumber Total_Lettuce
##   <chr>          <dbl>         <dbl>
## 1 15-03-13        4904           176
## 2 15-03-14        4904           182
## 3 15-03-15        4904           347
## 4 15-03-16        4904           400
## 5 15-03-17        4904           406
## 6 15-03-18        4904           382
```

Saving the data processed in a new excel sheet.

```r
library(openxlsx)
```

```
## Warning: package 'openxlsx' was built under R version 4.3.3
```

```r
# Assuming 'total_lettuce_per_transaction' is your data frame
write.xlsx(total_lettuce_per_transaction, "total_lettuce_per_transaction.xlsx")
```

This step ensures that the 'date' column is treated as a Date object, making it easier to work with time series data.

```r
total_lettuce_per_transaction$date <- as.Date(total_lettuce_per_transaction$date, format="%y-%m-%d")
```

Total lettuce across all stores: we are making a chart for total lettuce used across all stores over the entire time frame. This helps us understand the trends in the lettuce usage quantities over the time period.

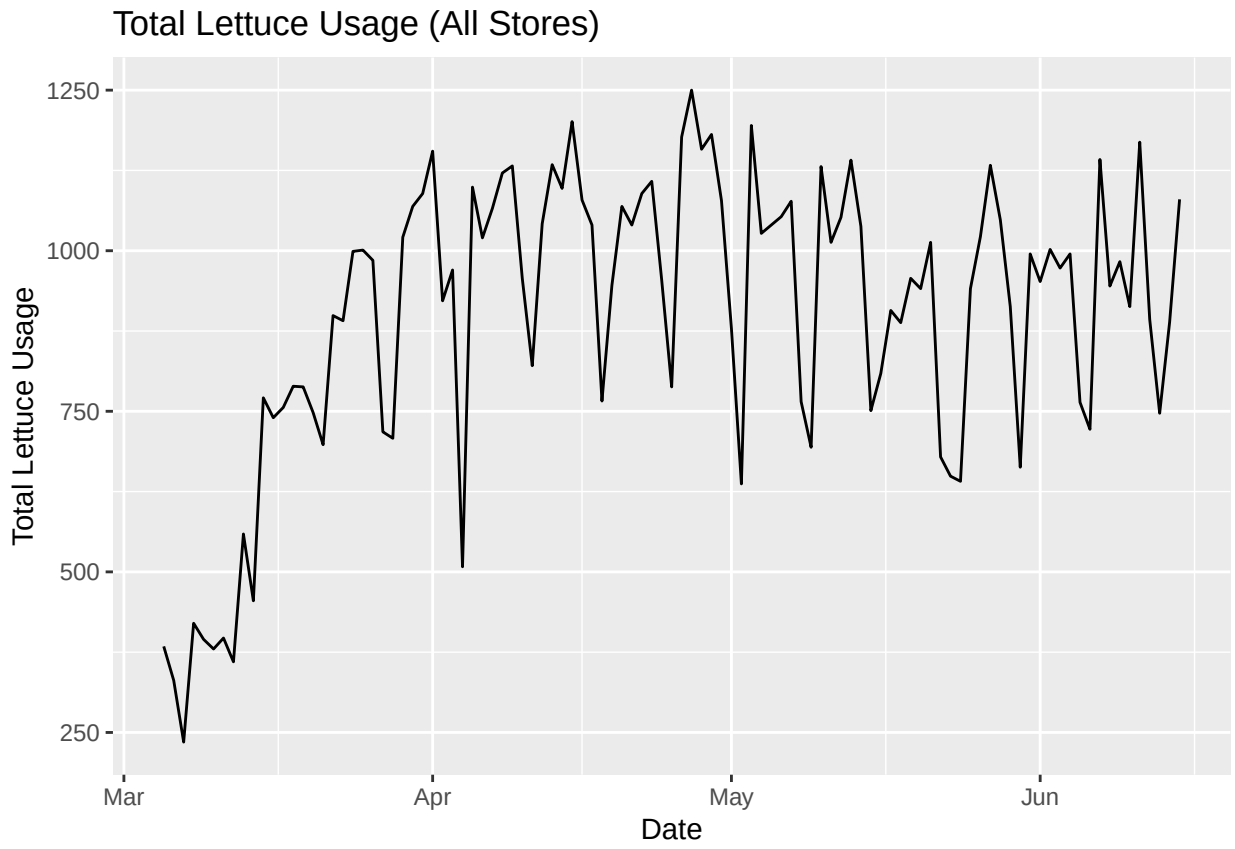
```r
library(zoo)
```

```
## Warning: package 'zoo' was built under R version 4.3.2
```

```
##
## Attaching package: 'zoo'
```

```
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```r
# Aggregate lettuce usage for each date
agg_data <- total_lettuce_per_transaction %>%
  group_by(date) %>%
  summarise(Total_Lettuce = sum(Total_Lettuce))

# Create a zoo object
lettuce_zoo <- zoo(agg_data$Total_Lettuce, as.Date(agg_data$date))

# Plot the time series
autoplot(lettuce_zoo) +
  labs(title = "Total Lettuce Usage (All Stores)",
       x = "Date",
       y = "Total Lettuce Usage")
```

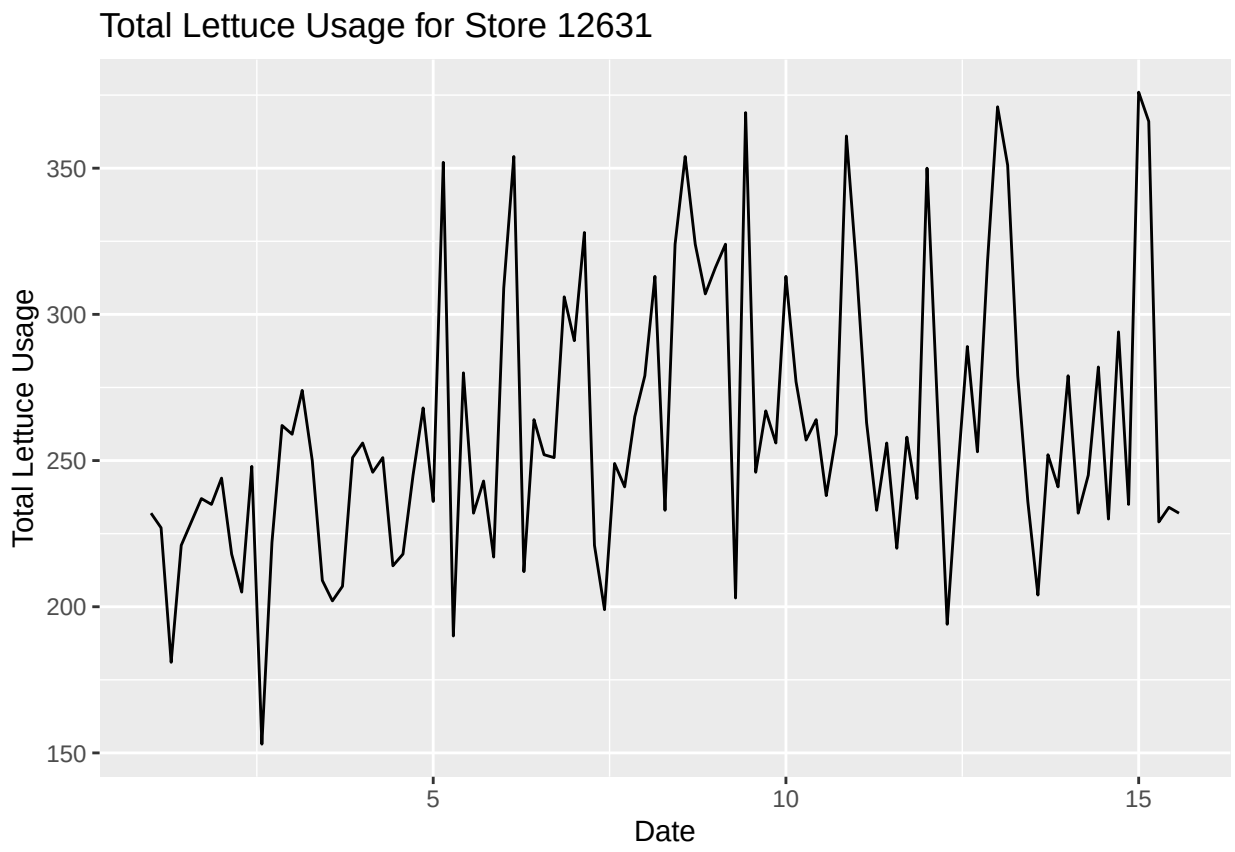


Store: 12631

In this analysis, we focused on a specific store, identified by the store number 12631. For our targeted store, we filtered the data to isolate lettuce-related transactions, creating a time series object to capture the daily total lettuce usage. The time series plot visualizes the fluctuations in lettuce consumption over time, providing insights into potential patterns or trends.

This approach allows us to gain a deeper understanding of the store's lettuce demand dynamics, a crucial factor for making informed inventory replenishment decisions.

```r
# Choose store number

store_number <- 12631

# Filter data for the chosen store
store_data_12631 <- total_lettuce_per_transaction %>% filter(StoreNumber == store_number)

# Create a zoo object
store_series <- zoo(store_data_12631$Total_Lettuce, as.Date(store_data_12631$date))

# Convert the 'date' column to Date format
store_data_12631$date <- as.Date(store_data_12631$date)

# Create a time series object
store_series_ts <- ts(store_data_12631$Total_Lettuce, frequency = 7)

# Plot the time series
autoplot(store_series_ts) +
  labs(title = paste("Total Lettuce Usage for Store", store_number),
       x = "Date",
       y = "Total Lettuce Usage")
```



We zoomed in on Store 12631, carefully dividing its lettuce usage data into two parts – an 80% chunk for training our forecasting models and a 20% slice to validate their accuracy. This split, respecting the chronological order of our data, sets the stage for effective model training and evaluation.

```r
split_proportion <- 0.8

# Function to split data for a specific store with window adjustment
split_data <- function(store_data, split_proportion) {
  split_index <- round(length(store_data) * split_proportion)
  train_data <- window(store_data, end = time(store_data)[split_index])
  test_data <- window(store_data, start = time(store_data)[split_index + 1])
  return(list(train_data = train_data, test_data = test_data))
}

# Split data for each store with window adjustment
split_12631 <- split_data(store_series_ts, split_proportion)

# Training Set (80%)
training_set_12631_ts <- split_12631$train_data

# Validation Set (20%)
validation_set_12631 <- split_12631$test_data
```

Part 1) ARIMA Model: I performed stationarity tests using the Augmented Dickey-Fuller (ADF), Phillips-Perron (PP), and Kwiatkowski-Phillips-Schmidt-Shin (KPSS) tests. Additionally, ndiffs and nsdiffs functions are used to determine the number of non-seasonal and seasonal differences needed to make the time series stationary, respectively.

```r
#ARIMA identification and order determination
adf.test(training_set_12631_ts)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  training_set_12631_ts
## Dickey-Fuller = -3.9522, Lag order = 4, p-value = 0.01598
## alternative hypothesis: stationary
```

```r
pp.test(training_set_12631_ts)
```

```
## Warning in pp.test(training_set_12631_ts): p-value smaller than printed p-value
```

```
##
##  Phillips-Perron Unit Root Test
##
## data:  training_set_12631_ts
## Dickey-Fuller Z(alpha) = -76.653, Truncation lag parameter = 3, p-value
## = 0.01
## alternative hypothesis: stationary
```

```r
kpss.test(training_set_12631_ts)
```

```
## Warning in kpss.test(training_set_12631_ts): p-value smaller than printed
## p-value
```

```
##
##  KPSS Test for Level Stationarity
##
## data:  training_set_12631_ts
## KPSS Level = 0.97892, Truncation lag parameter = 3, p-value = 0.01
```

```
# seasonal differencing
ndiffs(training_set_12631_ts)
```

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

```
nsdiffs(training_set_12631_ts)
```

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

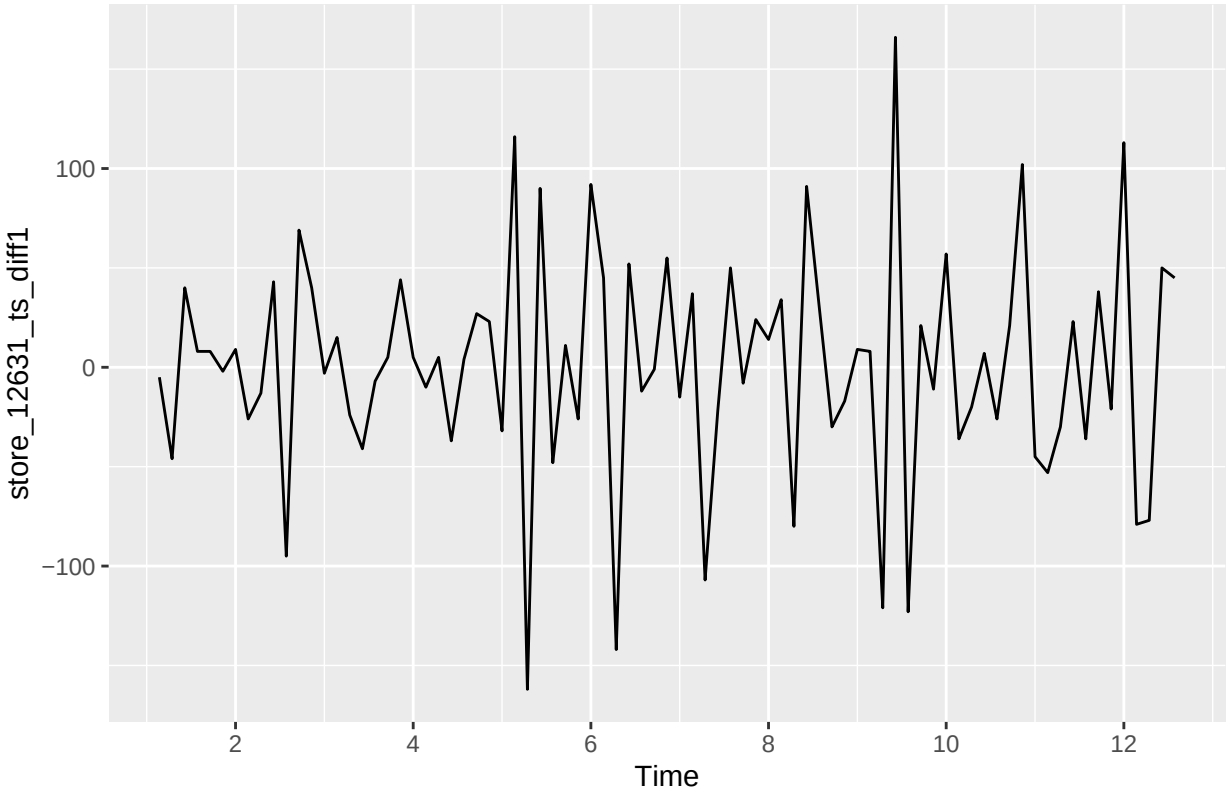
Key insights:

1) The ADF test suggests that the time series is stationary with a confidence level of 0.05. The PP test also indicates stationarity.
2) The KPSS test suggests non-stationarity at level, complementing the ADF and PP tests.
3) One non-seasonal difference is needed for stationarity.

In summary, based on these tests, the time series appears to be stationary after one non-seasonal difference.

Hence, we carry out differencing and plot our resulted data:

```
# ARIMA identification and order determination
store_12631_ts_diff1 <- diff(training_set_12631_ts, differences = 1)
autoplot(store_12631_ts_diff1)
```



Key insight: After differencing, data appears to be stationary now.

Now we recheck the required results through the same tests again:

```
#Now again check for all three tests
adf.test(store_12631_ts_diff1)
```

```
## Warning in adf.test(store_12631_ts_diff1): p-value smaller than printed p-value

##
##  Augmented Dickey-Fuller Test
##
## data:  store_12631_ts_diff1
## Dickey-Fuller = -7.1694, Lag order = 4, p-value = 0.01
## alternative hypothesis: stationary
```

```
pp.test(store_12631_ts_diff1)
```

```
## Warning in pp.test(store_12631_ts_diff1): p-value smaller than printed p-value

##
##  Phillips-Perron Unit Root Test
##
## data:  store_12631_ts_diff1
## Dickey-Fuller Z(alpha) = -104.26, Truncation lag parameter = 3, p-value
## = 0.01
## alternative hypothesis: stationary
```

```r
kpss.test(store_12631_ts_diff1)
```

```
## Warning in kpss.test(store_12631_ts_diff1): p-value greater than printed
## p-value
```

```
##
##  KPSS Test for Level Stationarity
##
## data:  store_12631_ts_diff1
## KPSS Level = 0.024078, Truncation lag parameter = 3, p-value = 0.1
```

```r
# seasonal differencing
ndiffs(store_12631_ts_diff1)
```

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

```r
nsdiffs(store_12631_ts_diff1)
```

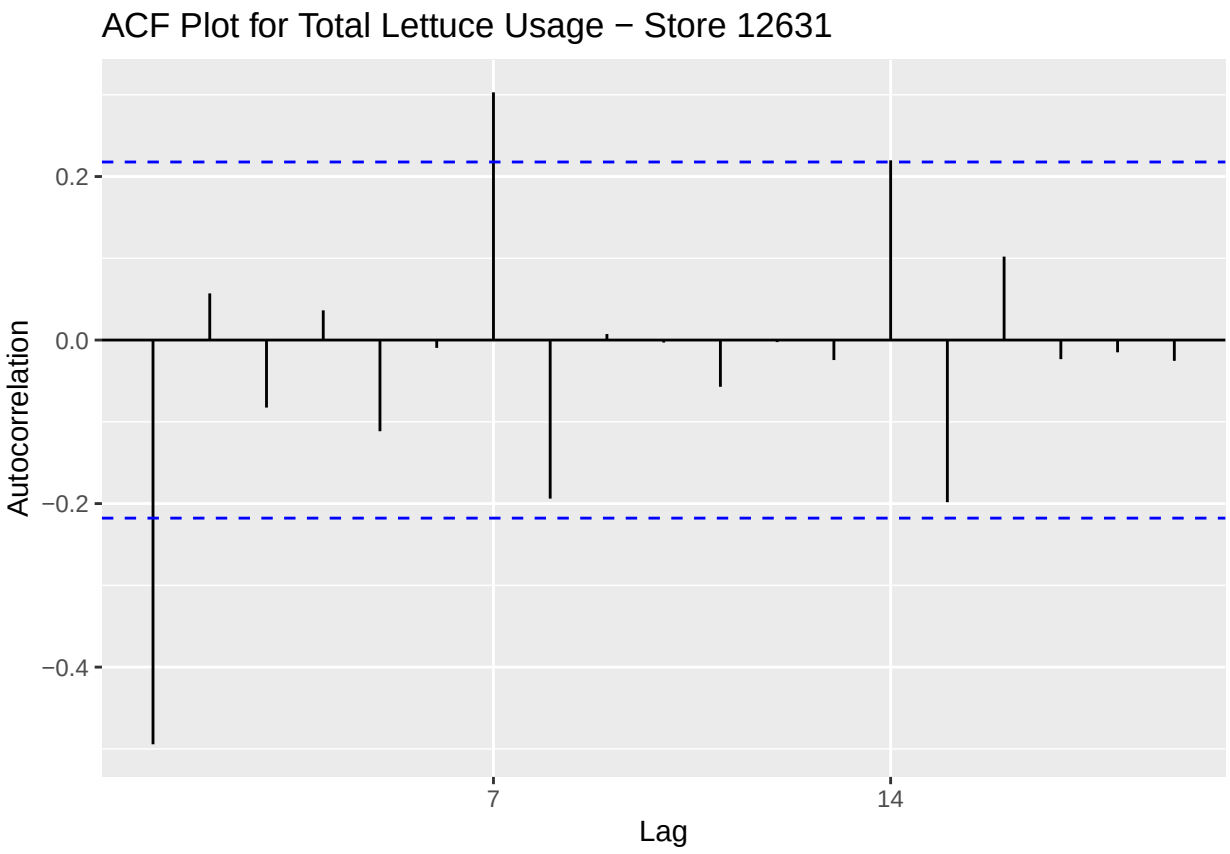
```
## [1] 0
```

The differenced time series, obtained through seasonal differencing, exhibits strong evidence of stationarity based on Augmented Dickey-Fuller and Phillips-Perron tests, supporting its suitability for further modeling and forecasting.

To further solidfy the selection of appropriate ARIMA parameters for our time series, we turn to the analysis of the autocorrelation function (ACF) and partial autocorrelation function (PACF). The visual examination of ACF and PACF plots aids in determining the optimal order for the ARIMA model, facilitating a more accurate and effective forecasting approach.

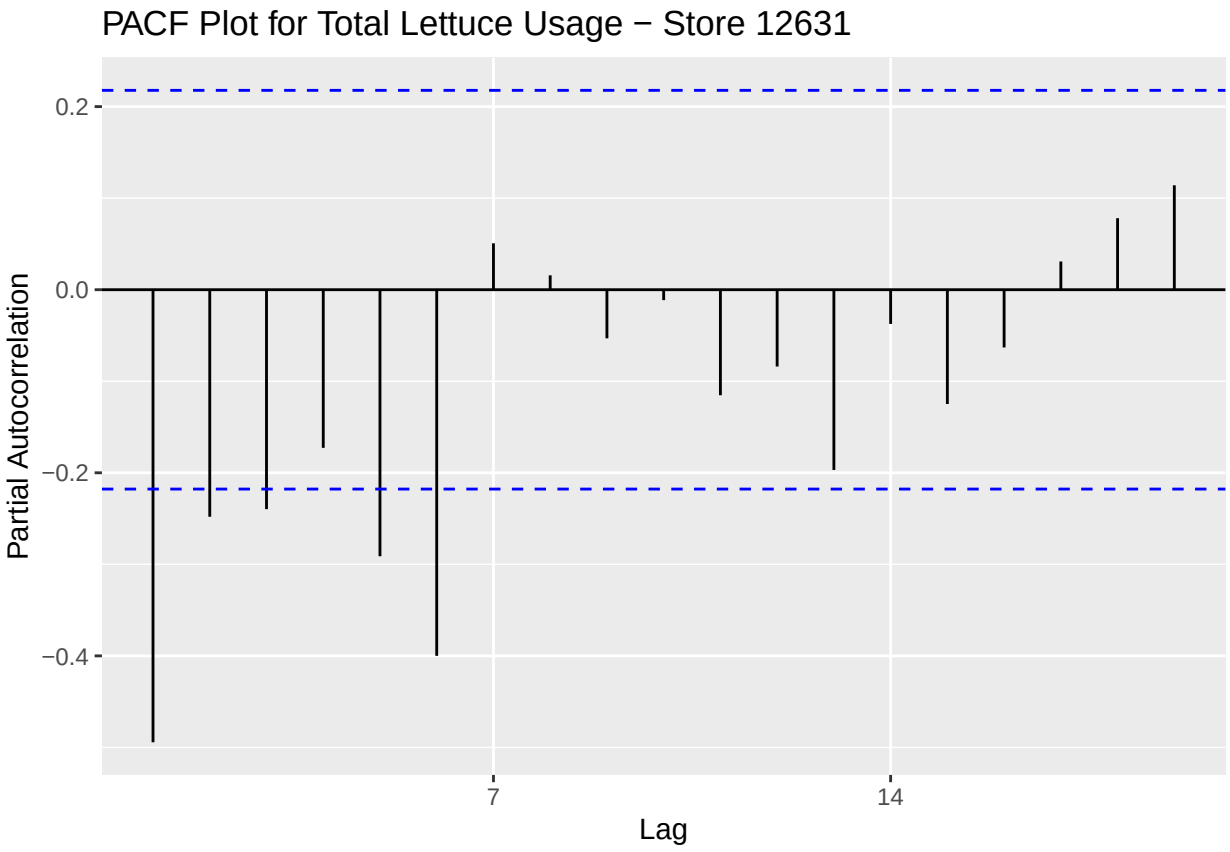
ACF Plot:

```r
# Determine p and q
ggAcf(store_12631_ts_diff1) +
  labs(title = "ACF Plot for Total Lettuce Usage - Store 12631",
       x = "Lag",
       y = "Autocorrelation")
```



PACF plot:

```
ggPacf(store_12631_ts_diff1) +
  labs(title = "PACF Plot for Total Lettuce Usage - Store 12631",
       x = "Lag",
       y = "Partial Autocorrelation")
```



Model Selection:

The auto.arima function was used to explore various combinations of differencing and seasonal parameters, considering the Akaike Information Criterion (AIC) as the selection criterion. Upon analysing the AIC values, three models with the lowest AICs were chosen. Subsequently, we proceeded to fit these selected models on our training data to further evaluate their forecasting capabilities.

(The code line for generating the auto arima models 'auto_arima_result_12631' is commented because of the long series of output. However, it stays uncommented in the RMD file)

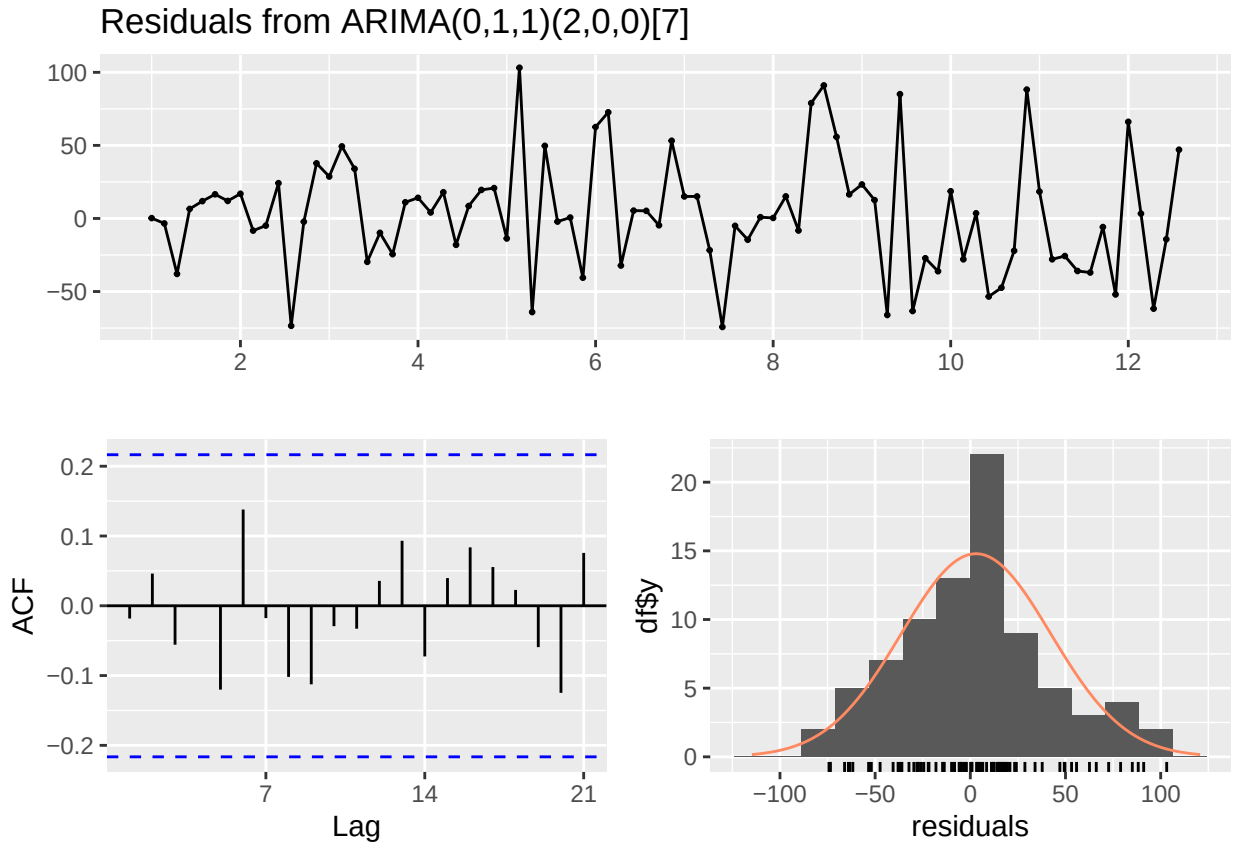
```
# Automatic ARIMA model selection
#auto_arima_result_12631 <- auto.arima(training_set_12631_ts, D = 0, d = 1,
#ic = 'aicc', trace = TRUE, stepwise = FALSE, approximation = FALSE)
#auto_arima_result_12631

#Best Models selection:
#ARIMA(0,1,1)(2,0,0)[7]: 836.1745
#ARIMA(0,1,1)(1,0,0)[7]: 836.8194
#ARIMA(0,1,1)(2,0,0)[7] with drift: 837.5495


# Fit the ARIMA model
arima_model_12631.m1 <- Arima(training_set_12631_ts, order = c(0, 1, 1),

seasonal = list(order = c(2, 0, 0), period = 7),
include.drift = FALSE)
```

```r
arima_model_12631.m2 <- Arima(training_set_12631_ts, order = c(0, 1, 1),

seasonal = list(order = c(1, 0, 0), period = 7),
include.drift = FALSE)

arima_model_12631.m3 <- Arima(training_set_12631_ts, order = c(0, 1, 1),

seasonal = list(order = c(2, 0, 0), period = 7),
include.drift = TRUE)
```

Residuals analysis:

To make sure that the models we selected effectively represent the temporal trends, we used residual analysis. For residuals, R's checkresiduals() function offers an extensive collection of statistical tests and diagnostic charts. In order to confirm the accuracy of our projections and to make appropriate judgments, this phase is essential.

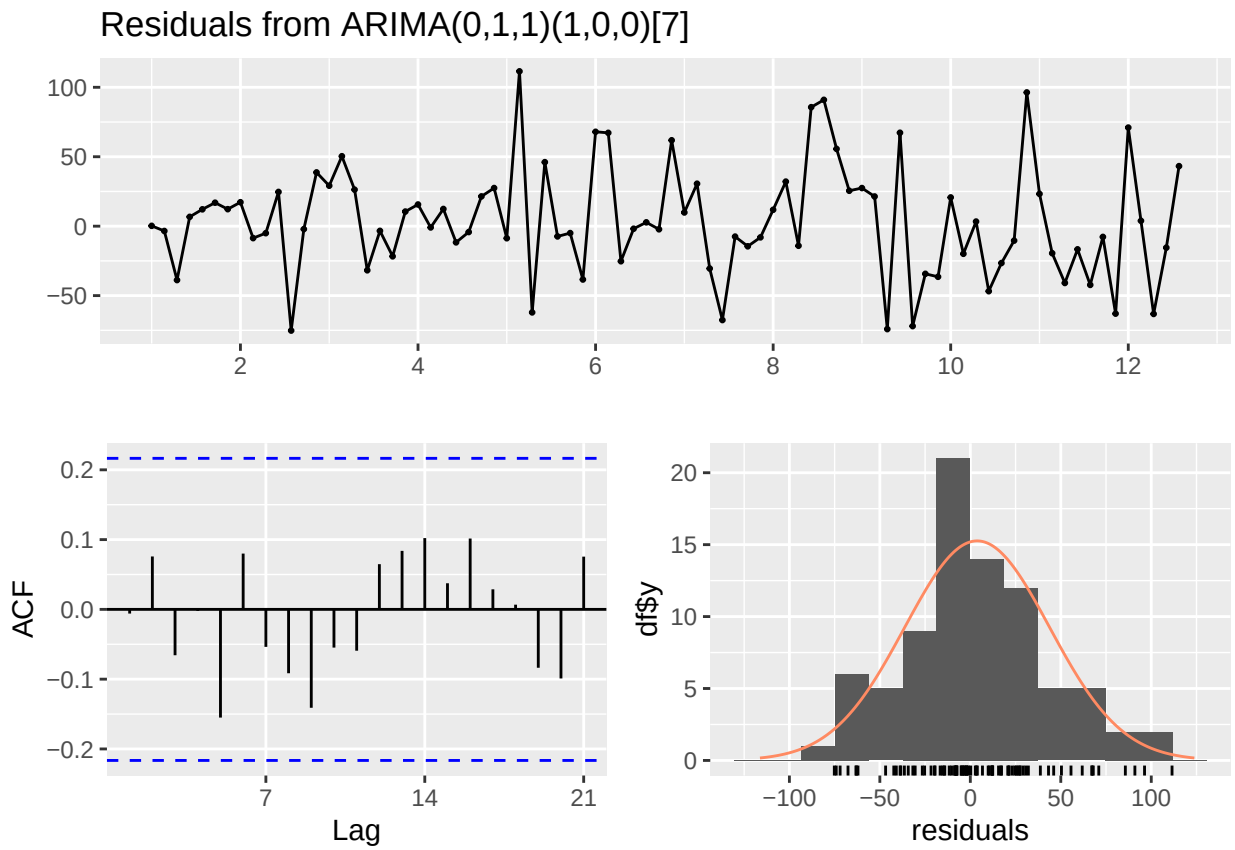
```r
# Residual analysis
checkresiduals(arima_model_12631.m1)
```




```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(0,1,1)(2,0,0)[7]
## Q* = 7.3998, df = 11, p-value = 0.7658
```

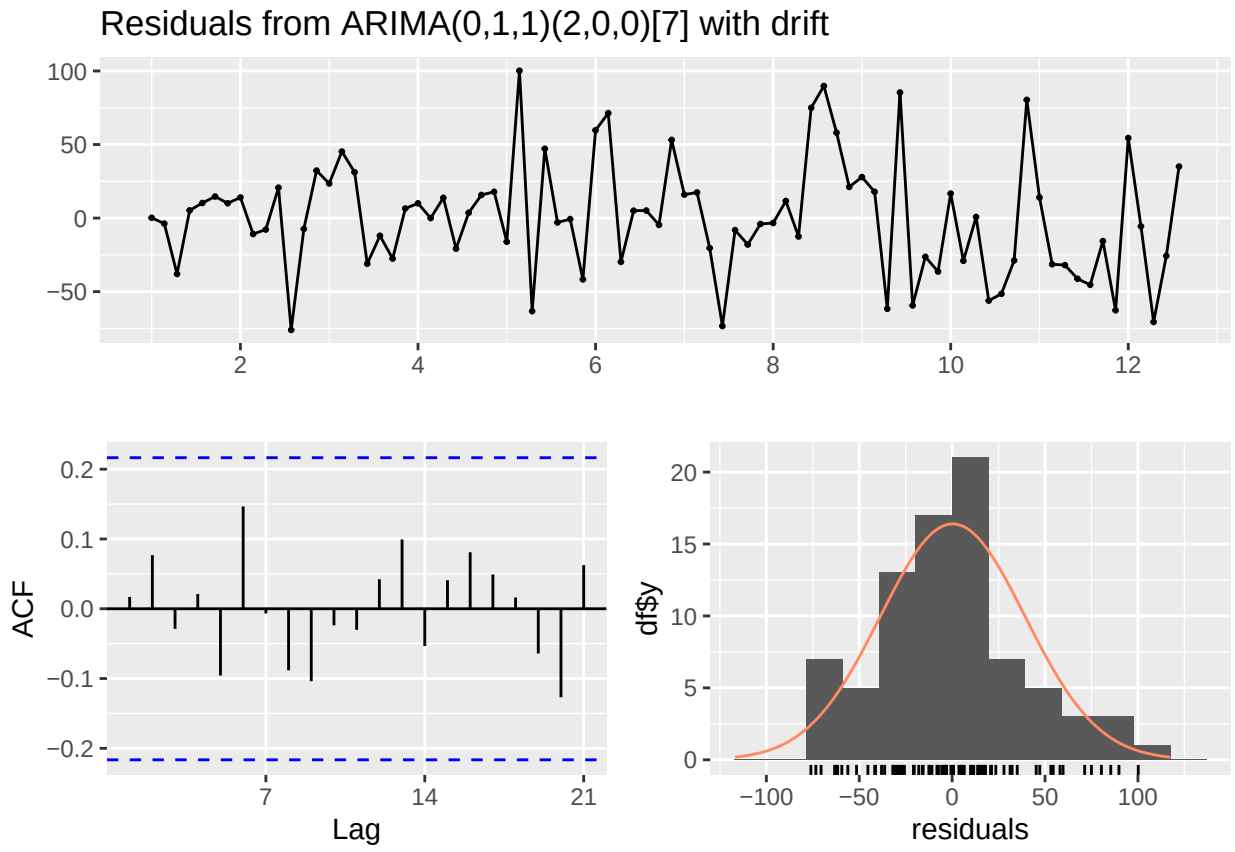
```
##
## Model df: 3.   Total lags used: 14
```

```
checkresiduals(arima_model_12631.m2)
```




```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(0,1,1)(1,0,0)[7]
## Q* = 9.3235, df = 12, p-value = 0.6751
##
## Model df: 2.   Total lags used: 14
```

```
checkresiduals(arima_model_12631.m3)
```




```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(0,1,1)(2,0,0)[7] with drift
## Q* = 6.7491, df = 11, p-value = 0.819
##
## Model df: 3.   Total lags used: 14
```

Forecasting and Accuracy:

Considering the fitted ARIMA models, we produce forecasts for the validation set in this stage. The validation set is a certain fraction of our time series data that was not used for model training. The 'accuracy' function, which offers a comprehensive set of parameters to evaluate the model's performance against the real data, is then used to determine how accurate these forecasts are. This analysis helps determine which ARIMA model is most likely to accurately estimate lettuce demand at Store 12631.

```
# Forecast using the fitted model
arima_forecast_12631.m1 <- forecast(arima_model_12631.m1, h = length(validation_set_12631))
arima_forecast_12631.m2 <- forecast(arima_model_12631.m2, h = length(validation_set_12631))
arima_forecast_12631.m3 <- forecast(arima_model_12631.m3, h = length(validation_set_12631))


# Model evaluation
accuracy(arima_forecast_12631.m1, validation_set_12631)
```

```
##                     ME      RMSE       MAE        MPE      MAPE      MASE
```

```
## Training set 2.975550 39.11431 29.74793 -0.8496374 11.53525 0.8365559
## Test set     9.605897 47.92302 38.56760  0.7354486 13.36942 1.0845783
##                     ACF1 Theil's U
## Training set -0.01825277        NA
## Test set      0.32505075 0.7748676
```

```
accuracy(arima_forecast_12631.m2, validation_set_12631)
```

```
##                     ME     RMSE      MAE         MPE     MAPE      MASE
## Training set 3.774626 39.94862 30.34739 -0.62574846 11.70352 0.8534136
## Test set     8.540539 50.68461 41.22940  0.08771721 14.42844 1.1594319
##                      ACF1 Theil's U
## Training set -0.005968094        NA
## Test set      0.330800123 0.8375307
```

```
accuracy(arima_forecast_12631.m3, validation_set_12631)
```

```
##                      ME     RMSE      MAE       MPE     MAPE      MASE
## Training set  0.3048809 38.81899 29.89376 -1.911078 11.66915 0.8406568
## Test set     -6.6921004 48.20361 42.95932 -5.473058 15.90192 1.2080798
##                    ACF1 Theil's U
## Training set 0.01692893        NA
## Test set     0.34583068 0.8051817
```

Key Insights:

1) With the lowest RMSE on the test sets, Model 1 performs better than the others according to the evaluation metrics, indicating that it is capable of forecasting lettuce demand at Store 12631 with greater accuracy.

2) Comparing this to models m2 and m3, a smaller RMSE demonstrates better performance in minimizing the prediction error.

3) The smaller RMSE values indicate that Model 1's forecasts align more closely with the actual values, making it the preferred choice for lettuce demand prediction in this context.

Fitting the best ARIMA Model on the entire data set:

The selected ARIMA(0,1,1)(2,0,0)[7] model, which has been determined to be the best-performing model on the training and validation sets, is being fitted to the whole dataset for Store 12631 in this stage of development. Establishing a definitive forecast for the subsequent 14 days is the ultimate objective.

```
#now fit the first best model on our entire dataset
arima_model_12631_final <- Arima(store_series_ts, order = c(0, 1, 1),
seasonal = list(order = c(2, 0, 0), period = 7), include.drift = FALSE)

# Final forecast
final_forecast_12631 <- forecast(arima_model_12631_final, h = 14)

# Extract point forecasts
forecast_values <- final_forecast_12631$mean

# Create forecast dates from 16/06/2015 to 29/06/2015
```

```r
forecast_dates <- seq(as.Date("2015-06-16"), by = "days", length.out = 14)

# Create a data frame with dates and point forecasts
final_forecast_data <- data.frame(date = forecast_dates, forecast = forecast_values)

# Print the final forecast data
print(final_forecast_data)
```

```
##          date forecast
## 1  2015-06-16 267.5754
## 2  2015-06-17 252.7773
## 3  2015-06-18 292.2427
## 4  2015-06-19 273.9320
## 5  2015-06-20 253.0645
## 6  2015-06-21 266.9463
## 7  2015-06-22 248.3694
## 8  2015-06-23 277.3772
## 9  2015-06-24 253.9750
## 10 2015-06-25 310.6632
## 11 2015-06-26 303.7647
## 12 2015-06-27 251.9276
## 13 2015-06-28 256.2538
## 14 2015-06-29 252.1068
```

Part 1 Conclusion:

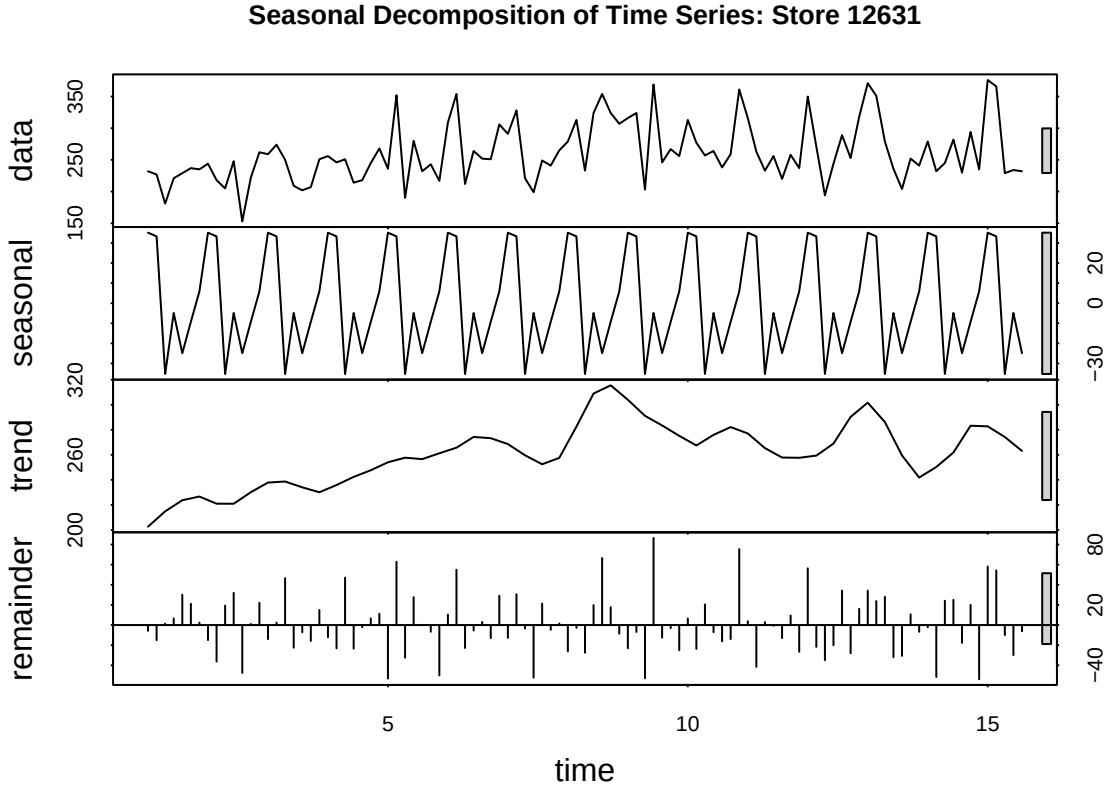
In summary, the model has been validated on historical data, and its accuracy assessed through metrics such as RMSE. We built a forecast for the next 14 days using this ARIMA model, which offers insightful information for inventory management.

Part 2) Holt Winters Model: Moving forward, we will extend our analysis by investigating the Holt-Winters model in order to determine the best method of forecasting the demand for lettuce in our dataset.

Seasonal Decomposition:

Decomposing a time series facilitates the process of recognizing and illustrating the underlying patterns in the data, including the general trend and recurring seasonal patterns. Determining the total variability in the time series is made easier by having a clear understanding of each component's contribution.

```r
plot(stl(store_series_ts, s.window = "period"), main="Seasonal Decomposition of Time Series: Store 1263
```



Key Insights:

1) The seasonal variations in the seasonal component plot get higher as the series continues on.
2) The fluctuations in the remainder/residual component plot appear to amplify with time, indicating that the irregular variations are also growing in size, which is inconsistent with constant variance.
3) The trend component seems to be comparatively flat or steady, suggesting that there is no clear trend and that it can be disregarded.

In summary, the increasing amplitude of both seasonal and irregular components over time, combined with the lack of a prominent trend, strongly indicates the need for a multiplicative model formulation.

Model selection:

I performed ETS (Exponential Smoothing State Space) forecasting using the training set for Store 12631. First, I used the ets() function without specifying a model (store_12631_ets_training2) to identify the best ETS model for the given data. I got (M,N,M) model as the best model in this case. I then used this model (store_12631_ets_training) on the training dataset for further analysis.

```
# ETS forecast based on the training set
store_12631_ets_training2 <- ets(training_set_12631_ts)

#(M,N,M) is our best ETS model.
# ETS MNM forecast based on the training set
store_12631_ets_training <- ets(training_set_12631_ts, model ='MNM')
store_12631_ets_training
```

```
## ETS(M,N,M)
##
## Call:
##  ets(y = training_set_12631_ts, model = "MNM")
##
##   Smoothing parameters:
##     alpha = 0.1549
##     gamma = 1e-04
##
##   Initial states:
##     l = 240.4624
##     s = 1.0391 0.9567 0.9337 1.0025 0.8566 1.1237
##            1.0877
##
##   sigma:  0.1438
##
##      AIC     AICc      BIC
## 962.0482 965.1467 986.1153
```

In-sample estimation error:

Here we are evaluating the in-sample estimation error of the ETS (Error, Trend, Seasonal) model using the accuracy() function, which provides metrics like Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and others.

```
# In-sample estimation error for ETS
in_sample_errors_ets <- accuracy(store_12631_ets_training)
in_sample_errors_ets
```

```
##                     ME     RMSE      MAE        MPE     MAPE      MASE
## Training set 2.025111 35.05182 26.63933 -0.8240822 10.28761 0.7491376
##                     ACF1
## Training set -0.0429736
```

In conclusion, these metrics provide insights into the performance of the ETS model on the training data.

Out-of-sample evaluation:

I am using the trained ETS model (store_12631_ets_training) to generate forecasts for the length of the validation set. Furthermore, I evaluate the accuracy of the ETS model's forecasts against the actual values in the validation set. This helps in understanding how well the ETS model performs on unseen data by comparing its forecasts with the actual values in the validation set, using a range of accuracy metrics.

```
#Out-of-sample evaluation for ETS
store_12631_ets_forecast <- forecast.ets(store_12631_ets_training, h = length(validation_set_12631))
store_12631_ets_forecast
```

```
##          Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
## 12.71429       254.3084 207.4426 301.1741 182.6334 325.9833
## 12.85714       276.1932 224.6747 327.7117 197.4025 354.9840
## 13.00000       289.1171 234.5467 343.6876 205.6589 372.5754
## 13.14286       298.6702 241.6415 355.6988 211.4524 385.8879
## 13.28571       227.7004 183.7289 271.6719 160.4518 294.9490
## 13.42857       266.4605 214.4321 318.4890 186.8899 346.0312
```

```
## 13.57143       248.1932 199.2044 297.1821 173.2713 323.1152
## 13.71429       254.3084 203.5772 305.0397 176.7217 331.8951
## 13.85714       276.1933 220.5214 331.8652 191.0505 361.3361
## 14.00000       289.1172 230.2443 347.9901 199.0789 379.1556
## 14.14286       298.6703 237.2423 360.0982 204.7243 392.6163
## 14.28571       227.7004 180.4084 274.9925 155.3735 300.0274
## 14.42857       266.4606 210.5843 322.3369 181.0051 351.9161
## 14.57143       248.1933 195.6547 300.7320 167.8424 328.5442
## 14.71429       254.3085 199.9741 308.6429 171.2112 337.4058
## 14.85714       276.1934 216.6442 335.7427 185.1207 367.2661
## 15.00000       289.1173 226.2221 352.0125 192.9274 385.3072
## 15.14286       298.6704 233.1239 364.2169 198.4256 398.9151
## 15.28571       227.7005 177.2958 278.1053 150.6131 304.7879
## 15.42857       266.4607 206.9728 325.9486 175.4818 357.4396
## 15.57143       248.1934 192.3188 304.0680 162.7406 333.6462
```

```
# Forecasting erros - ETS MODEL
out_of_sample_errors_ets <- accuracy(store_12631_ets_forecast,
                                     validation_set_12631)
out_of_sample_errors_ets
```

```
##                     ME     RMSE      MAE        MPE     MAPE      MASE
## Training set 2.025111 35.05182 26.63933 -0.8240822 10.28761 0.7491376
## Test set     7.479484 43.66539 35.88826  0.3521302 12.66273 1.0092313
##                    ACF1 Theil's U
## Training set -0.0429736        NA
## Test set      0.2390583 0.7300052
```

Building the MNM model on the entire data set:

The trained model is now employed to forecast the future values for the entire time series (store_series_ts). The model is trained on the entire dataset, and then we generate a forecast for the next 14 time points (h = 14).

```
# ETS MNM forecast on the entire dataset
final_ets_model <- ets(store_series_ts, model ='MNM')
final_ets_forecast <- forecast.ets(final_ets_model, h = 14)

# Extract point forecasts
forecast_values_ets <- final_ets_forecast$mean

# Create forecast dates from 16/06/2015 to 29/06/2015
forecast_dates_ets <- seq(as.Date("2015-06-16"), by = "days", length.out = 14)

# Create a data frame with dates and point forecasts
final_forecast_data_ets <- data.frame(date = forecast_dates_ets, forecast = forecast_values_ets)

# Print the final forecast data
print(final_forecast_data_ets)
```

```
##         date forecast
## 1 2015-06-16 258.1965
## 2 2015-06-17 276.1955
```

```
## 3  2015-06-18 302.6475
## 4  2015-06-19 307.1361
## 5  2015-06-20 233.2254
## 6  2015-06-21 266.3445
## 7  2015-06-22 246.8641
## 8  2015-06-23 258.1966
## 9  2015-06-24 276.1956
## 10 2015-06-25 302.6476
## 11 2015-06-26 307.1361
## 12 2015-06-27 233.2255
## 13 2015-06-28 266.3446
## 14 2015-06-29 246.8641
```

Part 2 Conclusion:

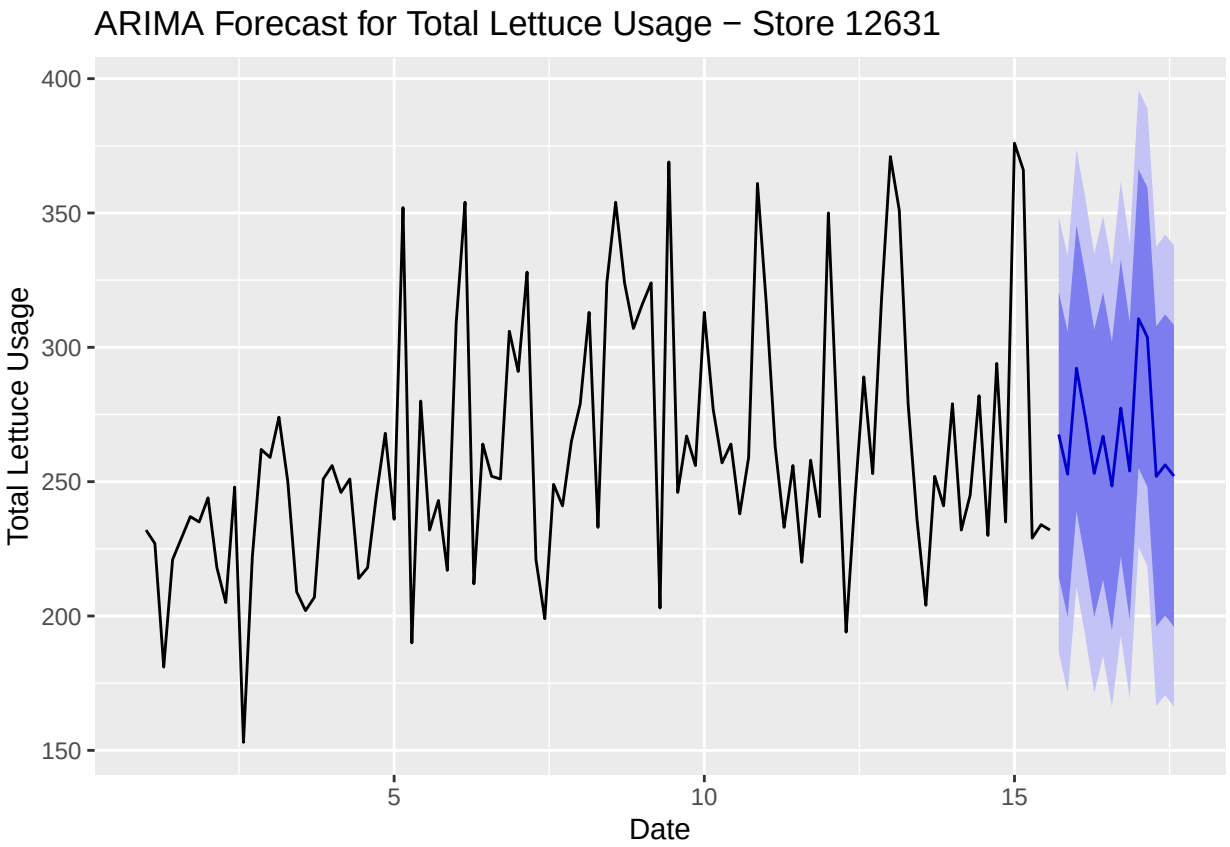
The resulting 'final_forecast_data_ets' contains the forecasted values and allows me to project future trends and seasonality based on the characteristics learned from the historical data.

Part 3 Comparison: Now, that we have both the Arima and the Holt winters model setup, we need to assess the forecasting performance of both the ARIMA and Holt-Winters models for a two-week timeframe.

To gain insights into the accuracy of each model's predictions, we initiated the examination by visualizing the forecasted values individually for ARIMA and Holt-Winters. This step allows us to observe the pattern and behavior of the forecasted series generated by each model.

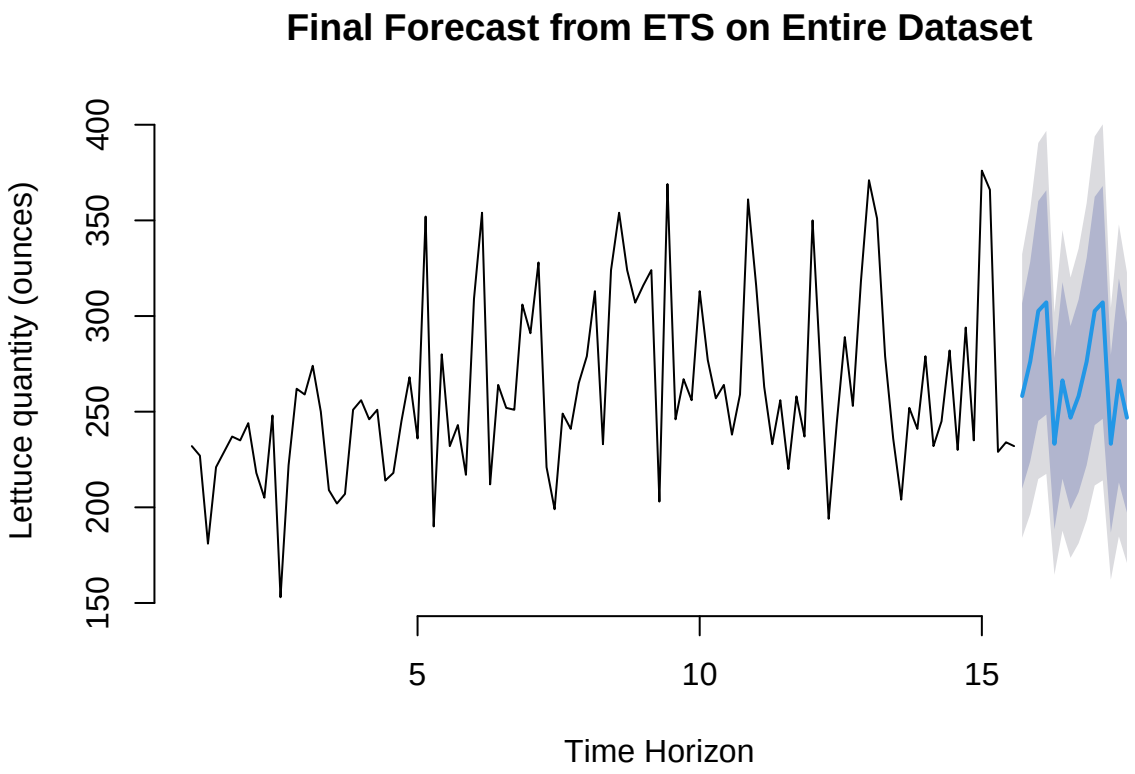
A) Plot the forecasts of the ARIMA Model:

```r
# Plot the forecast
autoplot(final_forecast_12631) +
  labs(title = "ARIMA Forecast for Total Lettuce Usage - Store 12631",
       x = "Date",
       y = "Total Lettuce Usage")
```



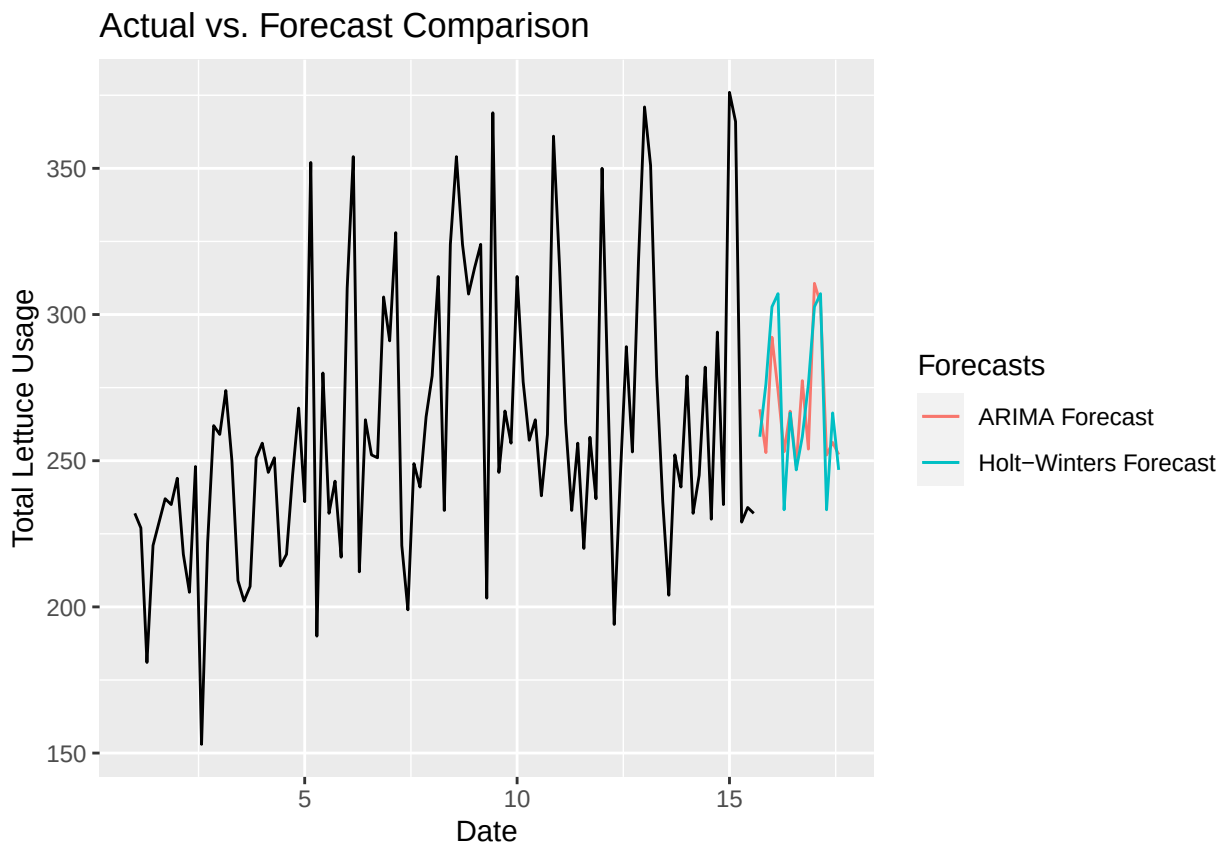
B) Plot the forecasts of the Holt Winter Model:

```
# Plot the final forecast
plot(final_ets_forecast, main = "Final Forecast from ETS on Entire Dataset", xlab="Time Horizon", ylab=
```



C) ARIMA vs HoltWinters Model Forecasts:

```
# Plot actual data
autoplot(store_series_ts) +
  labs(title = "Actual vs. Forecast Comparison",
       x = "Date",
       y = "Total Lettuce Usage") +
  autolayer(final_forecast_12631$mean, series = "ARIMA Forecast") +
  autolayer(final_ets_forecast$mean, series = "Holt-Winters Forecast") +
  guides(color = guide_legend(title = "Forecasts"))
```



Analysis of the Accuracies of both the Models:

Following the visual analysis of the ARIMA and Holt-Winters forecasts, we proceeded to a quantitative assessment using the Root Mean Squared Error (RMSE). RMSE is a widely utilized metric for evaluating the accuracy of forecasting models, providing a comprehensive measure of the differences between predicted and observed values. By calculating the RMSE for both the ARIMA and Holt-Winters models, we aim to quantify the level of accuracy each model achieves in capturing the actual values. A lower RMSE signifies better predictive performance, indicating that the model's forecasts closely align with the observed data. This numerical comparison aids in selecting the model that exhibits superior accuracy and reliability in forecasting the daily demand for lettuce in our dataset over the specified two-week period.

```
#RMSE of Arima Model
accuracy(arima_forecast_12631.m1, validation_set_12631)
```

```
##                     ME     RMSE      MAE        MPE     MAPE      MASE
## Training set 2.975550 39.11431 29.74793 -0.8496374 11.53525 0.8365559
## Test set     9.605897 47.92302 38.56760  0.7354486 13.36942 1.0845783
##                    ACF1 Theil's U
## Training set -0.01825277        NA
## Test set      0.32505075 0.7748676
```

```
#RMSE of ETS Model
out_of_sample_errors_ets
```

```
##                     ME     RMSE      MAE        MPE     MAPE      MASE
## Training set 2.025111 35.05182 26.63933 -0.8240822 10.28761 0.7491376
```

```
## Test set     7.479484 43.66539 35.88826  0.3521302 12.66273 1.0092313
##                     ACF1 Theil's U
## Training set -0.0429736        NA
## Test set      0.2390583 0.7300052
```

Key Insights:

RMSE Values	ARIMA	Holt Winters
Training data	39.11431	35.05182
Test data	47.92302	43.66539

ETS model, the MNM Model, has lower RMSE values for both the training and test sets compared to the ARIMA model. This suggests that the ETS model provides a better fit to the data in terms of forecasting accuracy. The lower RMSE values indicate that the ETS model's predictions are closer to the actual values, both in the training and test phases.

In summary, based on the RMSE values, the ETS model outperforms the ARIMA model in this specific analysis.

Final Forecasted Values:

Hence, the final predictions for the Store 12631 using the ETS - MNM model are:

```
# Print the final forecast data
print(final_forecast_data_ets)
```

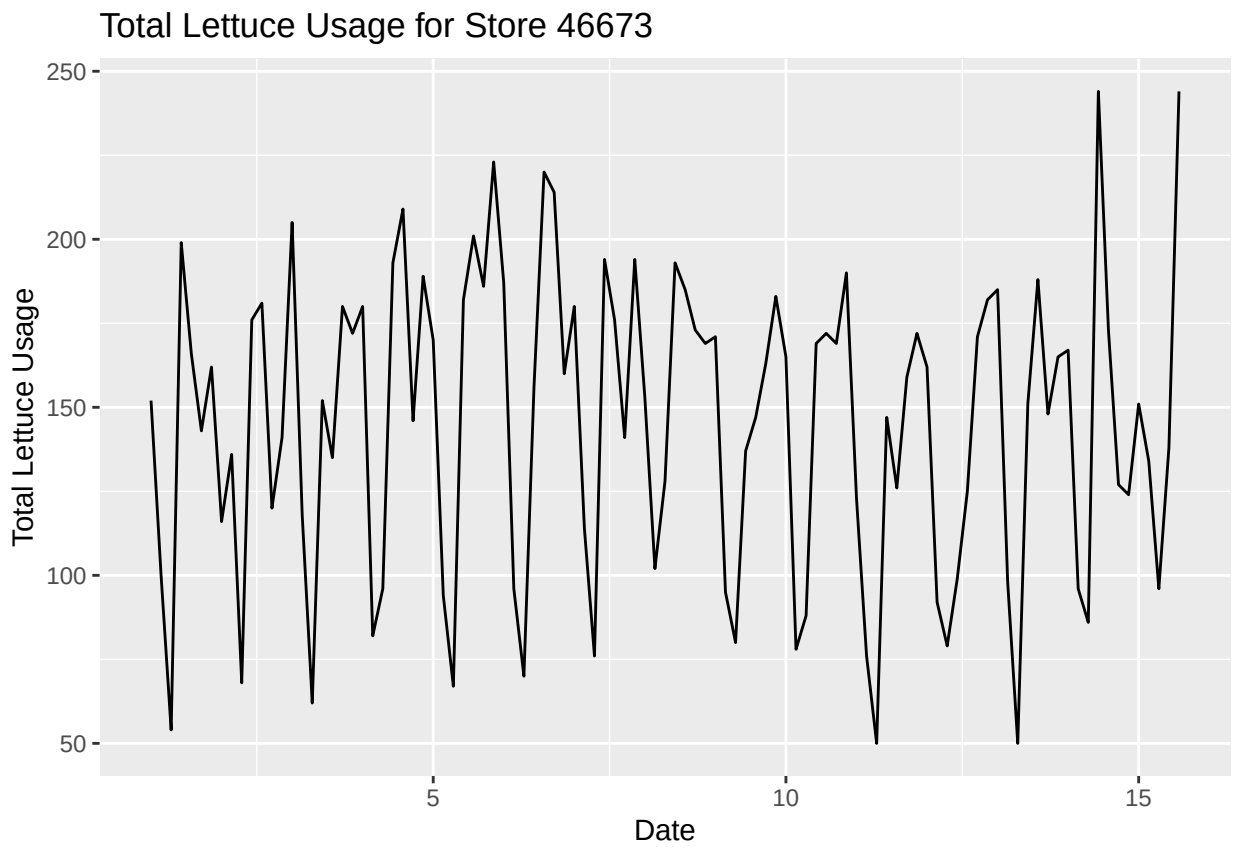
```
##          date forecast
## 1  2015-06-16 258.1965
## 2  2015-06-17 276.1955
## 3  2015-06-18 302.6475
## 4  2015-06-19 307.1361
## 5  2015-06-20 233.2254
## 6  2015-06-21 266.3445
## 7  2015-06-22 246.8641
## 8  2015-06-23 258.1966
## 9  2015-06-24 276.1956
## 10 2015-06-25 302.6476
## 11 2015-06-26 307.1361
## 12 2015-06-27 233.2255
## 13 2015-06-28 266.3446
## 14 2015-06-29 246.8641
```

Store: 46673

In this analysis, we focused on a specific store, identified by the store number 46673. For our targeted store, we filtered the data to isolate lettuce-related transactions, creating a time series object to capture the daily total lettuce usage. The time series plot visualizes the fluctuations in lettuce consumption over time, providing insights into potential patterns or trends.

This approach allows us to gain a deeper understanding of the store's lettuce demand dynamics, a crucial factor for making informed inventory replenishment decisions.

```r
# Choose store number
store_number <- 46673

# Filter data for the chosen store
store_data_46673 <- total_lettuce_per_transaction %>% filter(StoreNumber == store_number)

# Create a zoo object
store_series <- zoo(store_data_46673$Total_Lettuce, as.Date(store_data_46673$date))

# Create a time series object
store_series_ts <- ts(store_data_46673$Total_Lettuce, frequency = 7)

# Plot the time series
autoplot(store_series_ts) +
  labs(title = paste("Total Lettuce Usage for Store", store_number),
       x = "Date",
       y = "Total Lettuce Usage")
```



We zoomed in on Store 46673, carefully dividing its lettuce usage data into two parts – an 80% chunk for training our forecasting models and a 20% slice to validate their accuracy. This split, respecting the chronological order of our data, sets the stage for effective model training and evaluation.

```r
split_proportion <- 0.8

# Function to split data for a specific store with window adjustment
split_data <- function(store_data, split_proportion) {
```

```
  split_index <- round(length(store_data) * split_proportion)
  train_data <- window(store_data, end = time(store_data)[split_index])
  test_data <- window(store_data, start = time(store_data)[split_index + 1])
  return(list(train_data = train_data, test_data = test_data))
}

# Split data for each store with window adjustment
split_46673 <- split_data(store_series_ts, split_proportion)

# Training Set (80%)
training_set_46673_ts <- split_46673$train_data
# Validation Set (20%)
validation_set_46673 <- split_46673$test_data
```

Part 1) ARIMA Model: I performed stationarity tests using the Augmented Dickey-Fuller (ADF), Phillips-Perron (PP), and Kwiatkowski-Phillips-Schmidt-Shin (KPSS) tests. Additionally, ndiffs and nsdiffs functions are used to determine the number of non-seasonal and seasonal differences needed to make the time series stationary, respectively.

```
# ARIMA identification and order determination
adf.test(training_set_46673_ts)
```

```
## Warning in adf.test(training_set_46673_ts): p-value smaller than printed
## p-value
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  training_set_46673_ts
## Dickey-Fuller = -7.3051, Lag order = 4, p-value = 0.01
## alternative hypothesis: stationary
```

```
pp.test(training_set_46673_ts)
```

```
## Warning in pp.test(training_set_46673_ts): p-value smaller than printed p-value
```

```
##
##  Phillips-Perron Unit Root Test
##
## data:  training_set_46673_ts
## Dickey-Fuller Z(alpha) = -49.81, Truncation lag parameter = 3, p-value
## = 0.01
## alternative hypothesis: stationary
```

```
kpss.test(training_set_46673_ts)
```

```
## Warning in kpss.test(training_set_46673_ts): p-value greater than printed
## p-value
```

```
##
##  KPSS Test for Level Stationarity
##
## data:  training_set_46673_ts
## KPSS Level = 0.18882, Truncation lag parameter = 3, p-value = 0.1
```

```
# seasonal differencing
ndiffs(training_set_46673_ts)
```

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

```
nsdiffs(training_set_46673_ts)
```

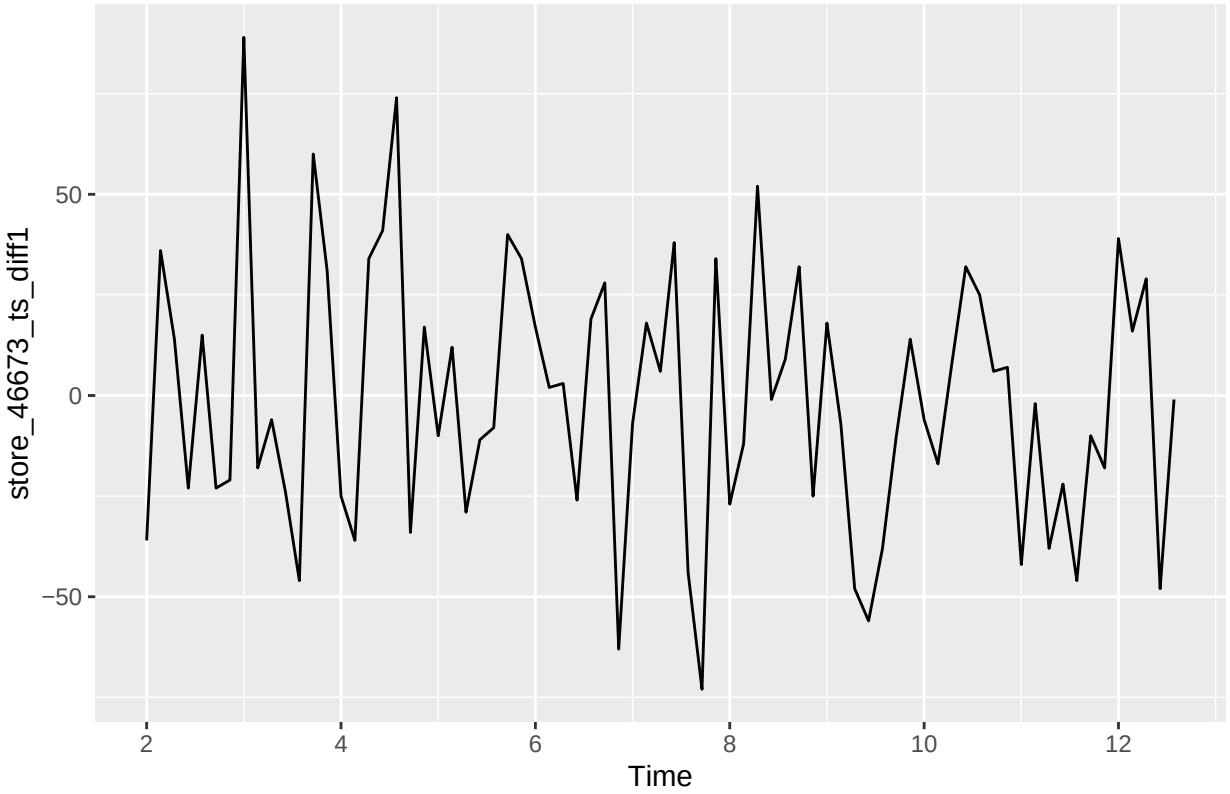
```
## [1] 1
```

Key insights:

1. The ADF test suggests that the time series is stationary with a confidence level of 0.05. The PP test also indicates stationarity.
2. One non-seasonal difference is needed for stationarity.

In summary, based on these tests, the time series appears to be stationary after one non-seasonal difference. Hence, we carry out differencing and plot our resulted data:

```
# ARIMA identification and order determination
store_46673_ts_diff1 <- diff(training_set_46673_ts, differences = 1, lag=7)
autoplot(store_46673_ts_diff1)
```



Key insight: After differencing, data appears to be stationary now.

Now we recheck the required results through the same tests again:

```
# ARIMA identification and order determination
adf.test(store_46673_ts_diff1)
```

```
## Warning in adf.test(store_46673_ts_diff1): p-value smaller than printed p-value

##
##  Augmented Dickey-Fuller Test
##
## data:  store_46673_ts_diff1
## Dickey-Fuller = -4.1898, Lag order = 4, p-value = 0.01
## alternative hypothesis: stationary
```

```
pp.test(store_46673_ts_diff1)
```

```
## Warning in pp.test(store_46673_ts_diff1): p-value smaller than printed p-value

##
##  Phillips-Perron Unit Root Test
##
## data:  store_46673_ts_diff1
## Dickey-Fuller Z(alpha) = -70.036, Truncation lag parameter = 3, p-value
## = 0.01
## alternative hypothesis: stationary
```

```r
kpss.test(store_46673_ts_diff1)
```

```
## Warning in kpss.test(store_46673_ts_diff1): p-value greater than printed
## p-value
```

```
##
##  KPSS Test for Level Stationarity
##
## data:  store_46673_ts_diff1
## KPSS Level = 0.21189, Truncation lag parameter = 3, p-value = 0.1
```

```r
# seasonal differencing
ndiffs(store_46673_ts_diff1)
```

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

```r
nsdiffs(store_46673_ts_diff1)
```

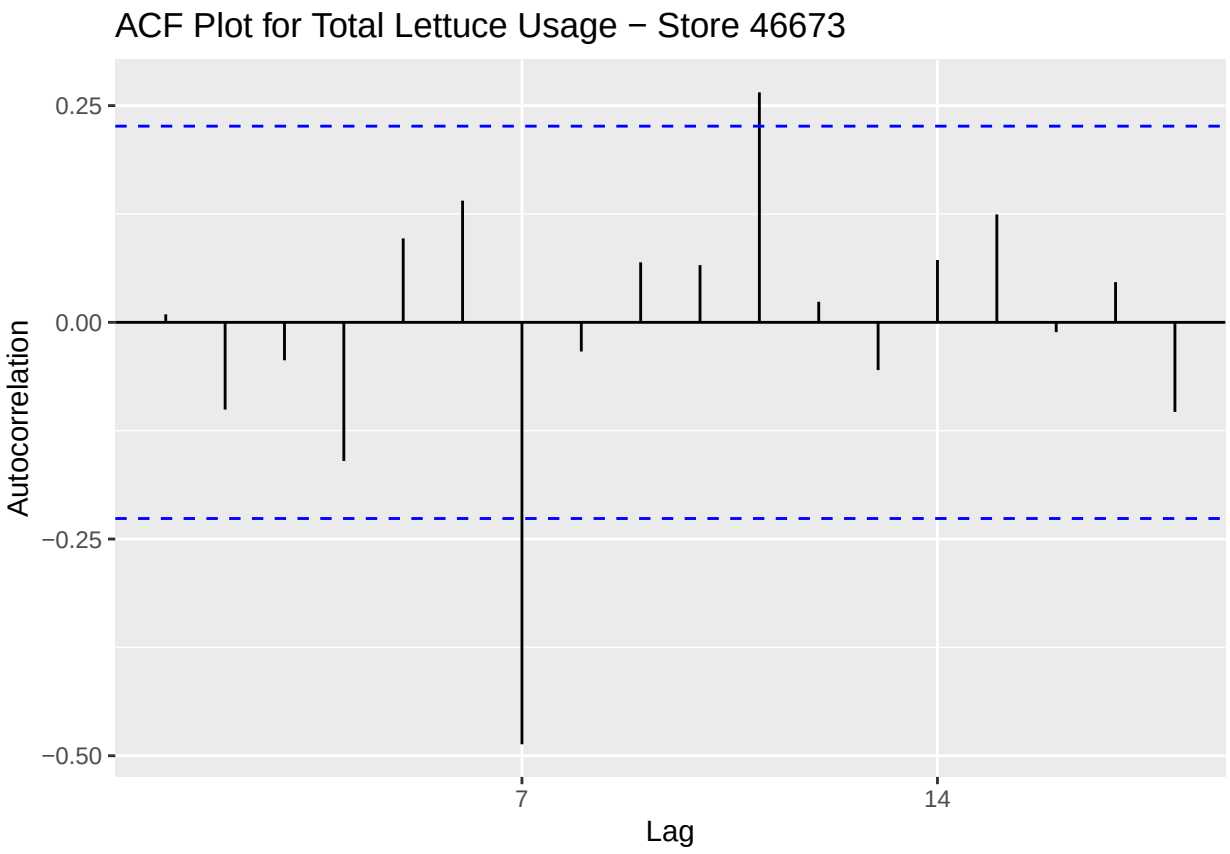
```
## [1] 0
```

The differenced time series, obtained through seasonal differencing, exhibits strong evidence of stationarity based on Augmented Dickey-Fuller and Phillips-Perron tests, supporting its suitability for further modeling and forecasting.

To further solidfy the selection of appropriate ARIMA parameters for our time series, we turn to the analysis of the autocorrelation function (ACF) and partial autocorrelation function (PACF). The visual examination of ACF and PACF plots aids in determining the optimal order for the ARIMA model, facilitating a more accurate and effective forecasting approach.

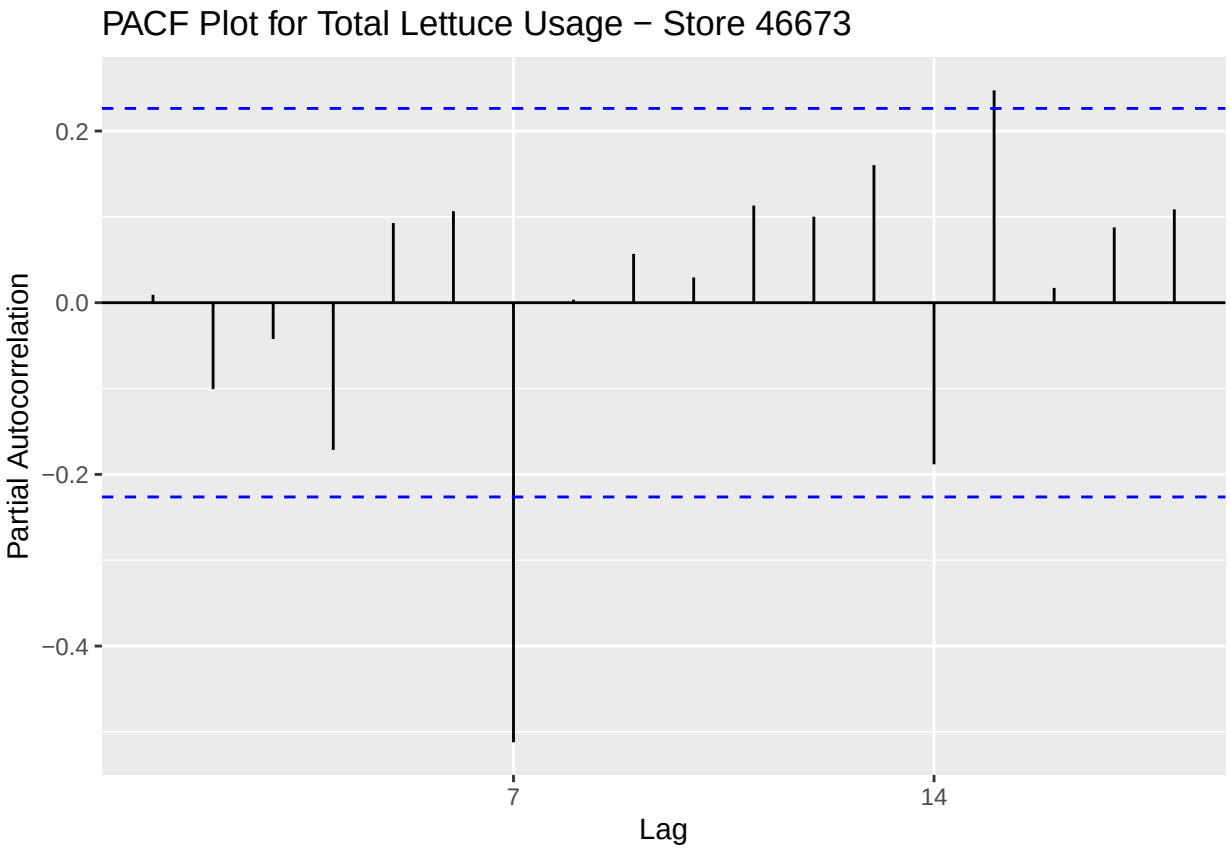
ACF Plot:

```r
# Determine p and q
ggAcf(store_46673_ts_diff1) +
  labs(title = "ACF Plot for Total Lettuce Usage - Store 46673",
       x = "Lag",
       y = "Autocorrelation")
```



PACF Plot:

```
ggPacf(store_46673_ts_diff1) +
  labs(title = "PACF Plot for Total Lettuce Usage - Store 46673",
       x = "Lag",
       y = "Partial Autocorrelation")
```



Model Selection:

The auto.arima function was used to explore various combinations of differencing and seasonal parameters, considering the Akaike Information Criterion (AIC) as the selection criterion. Upon analysing the AIC values, three models with the lowest AICs were chosen. Subsequently, we proceeded to fit these selected models on our training data to further evaluate their forecasting capabilities.

(The code line for generating the auto arima models 'auto_arima_result_46673' is commented because of the long series of output. However, it stays uncommented in the RMD file)

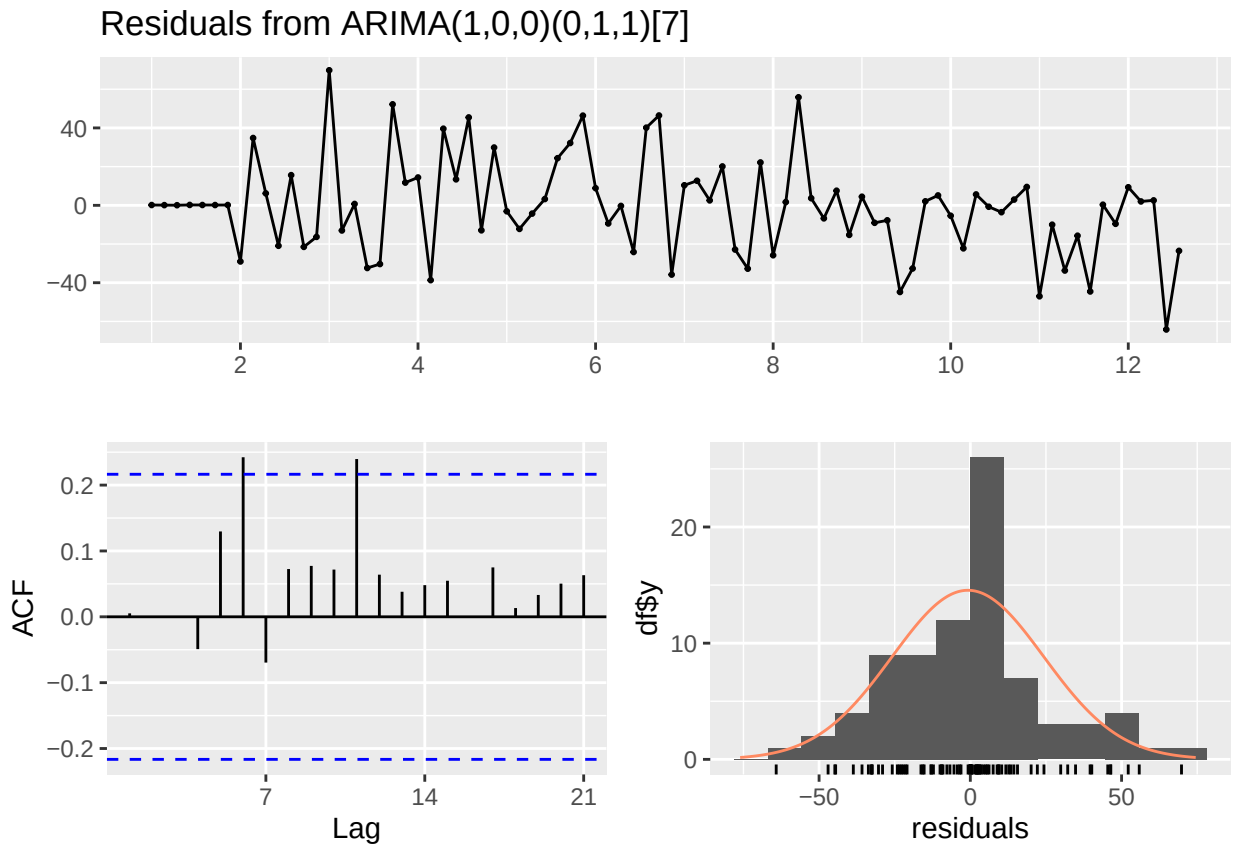
```
# Automatic ARIMA model selection
#auto_arima_result_46673 <- auto.arima(training_set_46673_ts, ic = 'aicc', D = 1, d = 0, approximation =

#auto_arima_result_46673

#best models:
#ARIMA(1,0,0)(0,1,1)[7]: 713.1908
#ARIMA(0,0,1)(0,1,1)[7]: 713.2317
#ARIMA(0,0,0)(0,1,1)[7]: 713.2449



# Fit the ARIMA model
arima_model_46673.m1 <- Arima(training_set_46673_ts, order = c(1, 0, 0),
seasonal = list(order = c(0, 1, 1), period = 7), include.drift = FALSE)
arima_model_46673.m2 <- Arima(training_set_46673_ts, order = c(0, 0, 1),
seasonal = list(order = c(0, 1, 1), period = 7), include.drift = FALSE)
```

```r
arima_model_46673.m3 <- Arima(training_set_46673_ts, order = c(0, 0, 0),
seasonal = list(order = c(0, 1, 1), period = 7), include.drift = FALSE)
```

Residuals analysis:

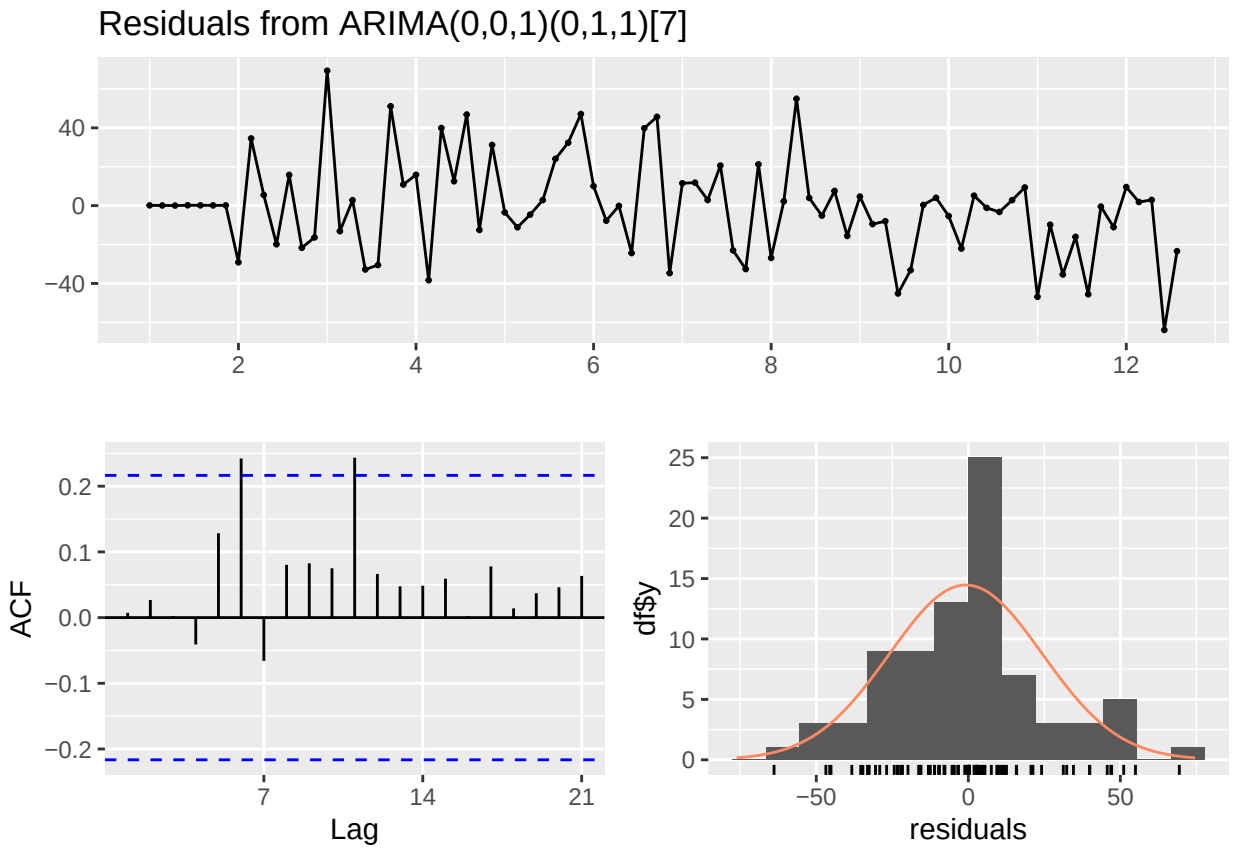
To make sure that the models we selected effectively represent the temporal trends, we used residual analysis. For residuals, R's checkresiduals() function offers an extensive collection of statistical tests and diagnostic charts. In order to confirm the accuracy of our projections and to make appropriate judgments, this phase is essential.

```r
# Residual analysis
checkresiduals(arima_model_46673.m1)
```



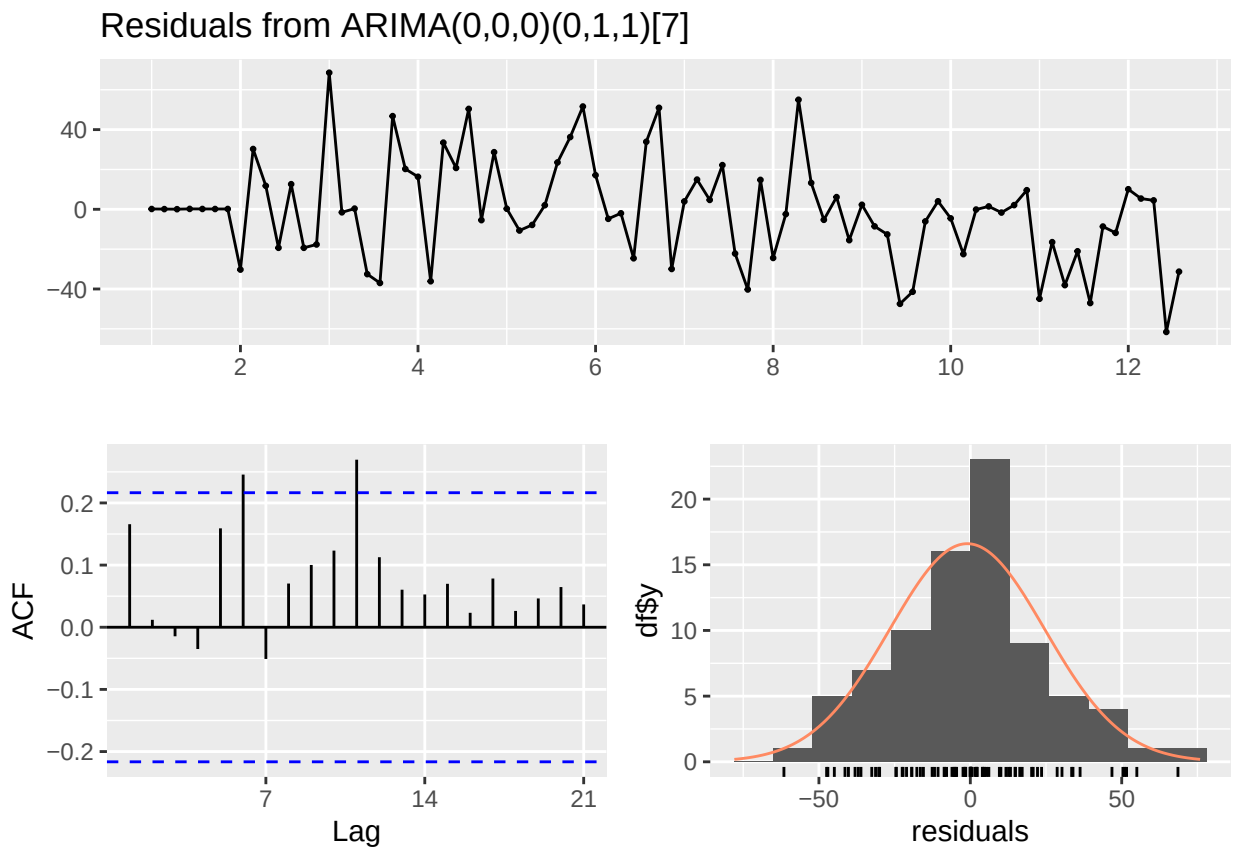

```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(1,0,0)(0,1,1)[7]
## Q* = 15.376, df = 12, p-value = 0.2215
##
## Model df: 2.   Total lags used: 14
```

```r
checkresiduals(arima_model_46673.m2)
```




```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(0,0,1)(0,1,1)[7]
## Q* = 15.824, df = 12, p-value = 0.1994
##
## Model df: 2.   Total lags used: 14
```

```
checkresiduals(arima_model_46673.m3)
```




```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(0,0,0)(0,1,1)[7]
## Q* = 22.255, df = 13, p-value = 0.05153
##
## Model df: 1.   Total lags used: 14
```

Forecasting and Accuracy:

Considering the fitted ARIMA models, we produce forecasts for the validation set in this stage. The validation set is a certain fraction of our time series data that was not used for model training. The 'accuracy' function, which offers a comprehensive set of parameters to evaluate the model's performance against the real data, is then used to determine how accurate these forecasts are. This analysis helps determine which ARIMA model is most likely to accurately estimate lettuce demand at Store 46673.

```
# Forecast using the fitted model
arima_forecast_46673.m1 <- forecast(arima_model_46673.m1, h = length(validation_set_46673))
arima_forecast_46673.m2 <- forecast(arima_model_46673.m2, h = length(validation_set_46673))
arima_forecast_46673.m3 <- forecast(arima_model_46673.m3, h = length(validation_set_46673))



# Model evaluation
accuracy(arima_forecast_46673.m1, validation_set_46673)
```

```
##                       ME     RMSE      MAE       MPE     MAPE      MASE
## Training set -0.7949027 24.95910 18.28022 -2.980306 13.47389 0.6903407
## Test set     11.7965214 38.00071 26.92164  3.587282 18.18053 1.0166781
##                     ACF1 Theil's U
## Training set 0.005226471        NA
## Test set     0.101673824 0.5088318
```

```
accuracy(arima_forecast_46673.m2, validation_set_46673)
```

```
##                       ME     RMSE      MAE       MPE     MAPE      MASE
## Training set -0.8097159 24.98450 18.30272 -2.989476 13.52350 0.6911902
## Test set     11.7236356 38.20382 26.96583  3.543133 18.21446 1.0183471
##                     ACF1 Theil's U
## Training set 0.007210826        NA
## Test set     0.103943913 0.5118469
```

```
accuracy(arima_forecast_46673.m3, validation_set_46673)
```

```
##                       ME     RMSE      MAE       MPE     MAPE      MASE      ACF1
## Training set -1.022294 25.49931 18.85525 -3.302016 14.01659 0.7120561 0.1657888
## Test set     13.030087 39.69353 27.85133  4.385599 18.68601 1.0517874 0.1144532
##              Theil's U
## Training set        NA
## Test set     0.5342211
```

```
#best rmse value is for arima_forecast_46673.m1 model
```

Key Insights:

1. With the lowest RMSE on the test sets, Model 1 performs better than the others according to the evaluation metrics, indicating that it is capable of forecasting lettuce demand at Store 46673 with greater accuracy.
2. Comparing this to models m2 and m3, a smaller RMSE demonstrates better performance in minimizing the prediction error.
3. The smaller RMSE values indicate that Model 1's forecasts align more closely with the actual values, making it the preferred choice for lettuce demand prediction in this context.

Fitting the best ARIMA Model on the entire data set:

The selected ARIMA(1,0,0)(0,1,1)[7] model, which has been determined to be the best-performing model on the training and validation sets, is being fitted to the whole dataset for Store 46673 in this stage of development. Establishing a definitive forecast for the subsequent 14 days is the ultimate objective.

```
#now fit the first best model on our entire dataset
arima_model_46673_final <- Arima(store_series_ts, order = c(1, 0, 0),
seasonal = list(order = c(0, 1, 1), period = 7), include.drift = FALSE)

# Final forecast
final_forecast_46673 <- forecast(arima_model_46673_final, h = 14)

final_forecast_46673
```

```
##          Point Forecast     Lo 80    Hi 80     Lo 95    Hi 95
## 15.71429      172.06567 137.22069 206.9107 118.77488 225.3565
## 15.85714      175.15870 139.83185 210.4855 121.13095 229.1864
## 16.00000      164.80336 129.53950 200.0672 110.87194 218.7348
## 16.14286      100.78962  65.52539 136.0539  46.85763 154.7216
## 16.28571       76.67597  41.41172 111.9402  22.74396 130.6080
## 16.42857      168.66758 133.40333 203.9318 114.73557 222.5996
## 16.57143      176.52969 141.26544 211.7939 122.59768 230.4617
## 16.71429      160.78112 125.44050 196.1217 106.73230 214.8299
## 16.85714      173.27133 137.93071 208.6120 119.22251 227.3201
## 17.00000      164.48770 129.22345 199.7519 110.55568 218.4197
## 17.14286      100.73682  65.47257 136.0011  46.80480 154.6688
## 17.28571       76.66714  41.40288 111.9314  22.73512 130.5992
## 17.42857      168.66610 133.40185 203.9304 114.73408 222.5981
## 17.57143      176.52944 141.26519 211.7937 122.59742 230.4615
```

```r
# Extract point forecasts
forecast_values_46673 <- final_forecast_46673$mean

# Create forecast dates from 16/06/2015 to 29/06/2015
forecast_dates_46673 <- seq(as.Date("2015-06-16"), by = "days", length.out = 14)

# Create a data frame with dates and point forecasts
final_forecast_data_46673 <- data.frame(date = forecast_dates_46673,
forecast = forecast_values_46673)

# Print the final forecast data
print(final_forecast_data_46673)
```

```
##          date  forecast
## 1  2015-06-16 172.06567
## 2  2015-06-17 175.15870
## 3  2015-06-18 164.80336
## 4  2015-06-19 100.78962
## 5  2015-06-20  76.67597
## 6  2015-06-21 168.66758
## 7  2015-06-22 176.52969
## 8  2015-06-23 160.78112
## 9  2015-06-24 173.27133
## 10 2015-06-25 164.48770
## 11 2015-06-26 100.73682
## 12 2015-06-27  76.66714
## 13 2015-06-28 168.66610
## 14 2015-06-29 176.52944
```

Part 1 Conclusion:

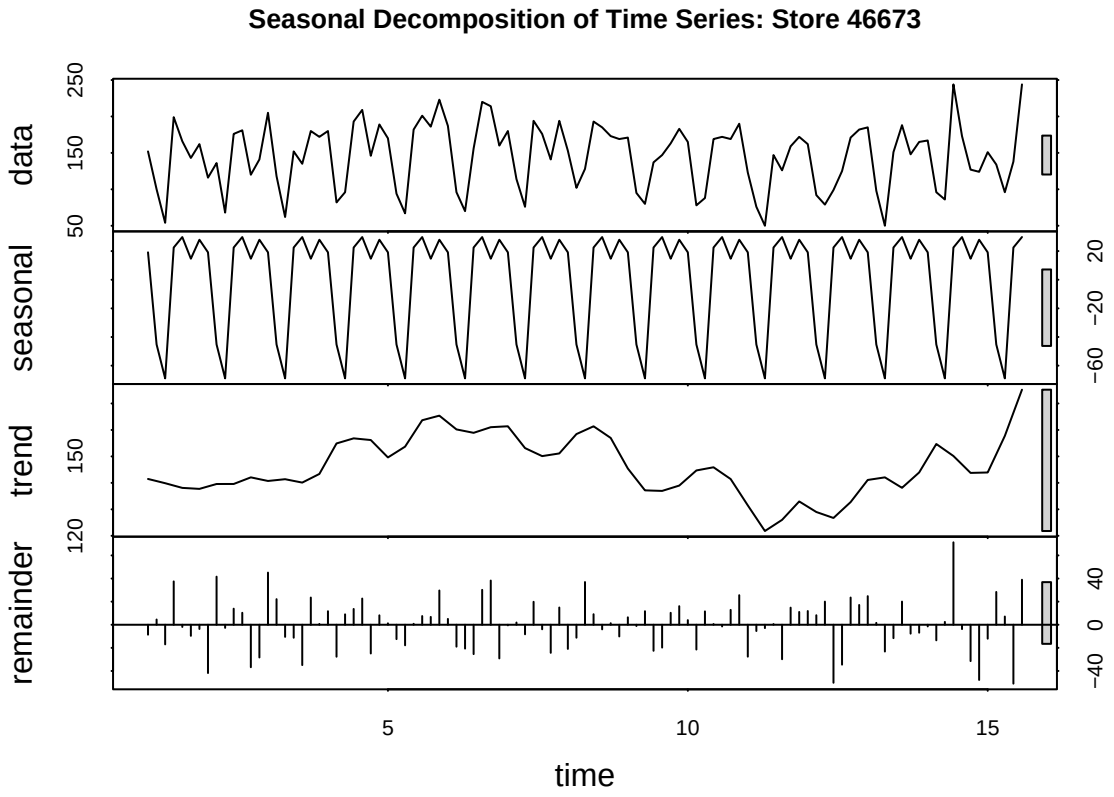
In summary, the model has been validated on historical data, and its accuracy assessed through metrics such as RMSE. We built a forecast for the next 14 days using this ARIMA model, which offers insightful information for inventory management.

Part 2) Holt Winters Model: Moving forward, we will extend our analysis by investigating the Holt-Winters model in order to determine the best method of forecasting the demand for lettuce in our dataset.

Seasonal Decomposition:

Decomposing a time series facilitates the process of recognizing and illustrating the underlying patterns in the data, including the general trend and recurring seasonal patterns. Determining the total variability in the time series is made easier by having a clear understanding of each component's contribution.

```
plot(stl(store_series_ts, s.window = "period"),
main="Seasonal Decomposition of Time Series: Store 46673", xaxt = "n")
```



Key Insights:

1) The trend component of the time-series can be disregarded because no particular, significant linear trend can be seen, according to the seasonal decomposition of the data.

2) Furthermore, it appears that the time series has an additive seasonal component, which will be required by the future forecast model to be developed in order to match the time series and provide future projections.

Model selection:

I performed ETS (Exponential Smoothing State Space) forecasting using the training set for Store 46673. First, I used the ets() function without specifying a model (store_46673_ets_training2) to identify the best ETS model for the given data. I got (A,N,A) model as the best model in this case. I then used this model (store_46673_ets_training) on the training dataset for further analysis.

```r
# ETS forecast based on the training set
store_46673_ets_training2 <- ets(training_set_46673_ts, model='ZZZ')

#ETS(A,N,A) is our best ETS model.
# ETS ANA forecast based on the training set
store_46673_ets_training <- ets(training_set_46673_ts, model='ANA')
store_46673_ets_training
```

```
## ETS(A,N,A)
##
## Call:
##  ets(y = training_set_46673_ts, model = "ANA")
##
##   Smoothing parameters:
##     alpha = 0.1223
##     gamma = 1e-04
##
##   Initial states:
##     l = 144.2834
##     s = 31.9515 15.5058 26.5879 24.4611 -70.2816 -46.5225
##           18.2978
##
##   sigma:  24.6121
##
##      AIC     AICc      BIC
## 897.1487 900.2473 921.2159
```

In-sample estimation error:

Here we are evaluating the in-sample estimation error of the ETS (Error, Trend, Seasonal) model using the accuracy() function, which provides metrics like Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and others.

```r
# In-sample estimation error for ETS
in_sample_errors_ets <- accuracy(store_46673_ets_training)
in_sample_errors_ets
```

```
##                      ME     RMSE      MAE       MPE     MAPE      MASE
## Training set -1.861661 23.22218 18.45667 -4.154372 14.30555 0.6970043
##                     ACF1
## Training set 0.05622874
```

In conclusion, these metrics provide insights into the performance of the ETS model on the training data.

Out-of-sample evaluation:

I am using the trained ETS model (store_46673_ets_training) to generate forecasts for the length of the validation set. Furthermore, I evaluate the accuracy of the ETS model's forecasts against the actual values in the validation set. This helps in understanding how well the ETS model performs on unseen data by comparing its forecasts with the actual values in the validation set, using a range of accuracy metrics.

```r
#Out-of-sample evaluation for ETS
store_46673_ets_forecast <- forecast.ets(store_46673_ets_training, h = length(validation_set_46673))
store_46673_ets_forecast
```

```
##          Point Forecast     Lo 80     Hi 80       Lo 95    Hi 95
## 12.71429      141.12073 109.57906 172.66240  92.8819147 189.3595
## 12.85714      157.56444 125.78780 189.34108 108.9662589 206.1626
## 13.00000      143.91068 111.90078 175.92057  94.9557710 192.8656
## 13.14286       79.09038  46.84892 111.33183  29.7813225 128.3994
## 13.28571       55.33355  22.86219  87.80492   5.6728805 104.9942
## 13.42857      150.07040 117.37073 182.77006 100.0605755 200.0802
## 13.57143      152.19993 119.27318 185.12667 101.8428128 202.5570
## 13.71429      141.12073 107.96882 174.27264  90.4192642 191.8222
## 13.85714      157.56444 124.18889 190.93999 106.5209434 208.6079
## 14.00000      143.91068 110.31298 177.50838  92.5274292 195.2939
## 14.14286       79.09038  45.27198 112.90877  27.3696055 130.8111
## 14.28571       55.33355  21.29590  89.37121   3.2774513 107.3897
## 14.42857      150.07040 115.81488 184.32591  97.6811082 202.4597
## 14.57143      152.19993 117.72758 186.67228  99.4790176 204.9208
## 14.71429      141.12073 106.43325 175.80821  88.0708104 194.1707
## 14.85714      157.56444 122.66316 192.46572 104.1875360 210.9413
## 15.00000      143.91068 108.79690 179.02446  90.2087823 197.6126
## 15.14286       79.09038  43.76537 114.41538  25.0654425 133.1153
## 15.28571       55.33355  19.79858  90.86853   0.9875039 109.6796
## 15.42857      150.07040 114.32669 185.81410  95.4051162 204.7357
## 15.57143      152.19993 116.24836 188.15149  97.2167512 207.1831
```

```
# Forecasting erros - ETS MODEL
out_of_sample_errors_ets <- accuracy(store_46673_ets_forecast, validation_set_46673)
out_of_sample_errors_ets
```

```
##                      ME     RMSE      MAE       MPE     MAPE      MASE
## Training set -1.861661 23.22218 18.45667 -4.154372 14.30555 0.6970043
## Test set     22.863318 38.16146 29.06228 13.661495 19.14726 1.0975182
##                    ACF1 Theil's U
## Training set 0.05622874        NA
## Test set     0.04582855 0.4904983
```

Building the ANA model on the entire data set:

The trained model is now employed to forecast the future values for the entire time series (store_series_ts). The model is trained on the entire dataset, and then we generate a forecast for the next 14 time points (h = 14).

```
# ETS ANA forecast on the entire dataset
final_ets_model <- ets(store_series_ts, model='ANA')
final_ets_forecast <- forecast.ets(final_ets_model, h = 14)

# Display the forecasted values
final_ets_forecast
```

```
##          Point Forecast     Lo 80    Hi 80     Lo 95    Hi 95
## 15.71429      159.79583 125.66254 193.9291 107.59347 211.9982
## 15.85714      173.08458 138.95130 207.2179 120.88223 225.2869
## 16.00000      164.56924 130.43595 198.7025 112.36688 216.7716
## 16.14286      101.27350  67.14021 135.4068  49.07114 153.4758
## 16.28571       76.91392  42.78064 111.0472  24.71157 129.1163
```

```
## 16.42857      170.12068 135.98740 204.2540 117.91833 222.3230
## 16.57143      175.76910 141.63581 209.9024 123.56674 227.9715
## 16.71429      159.79583 125.66254 193.9291 107.59347 211.9982
## 16.85714      173.08458 138.95130 207.2179 120.88223 225.2869
## 17.00000      164.56924 130.43595 198.7025 112.36688 216.7716
## 17.14286      101.27350  67.14021 135.4068  49.07114 153.4759
## 17.28571       76.91392  42.78064 111.0472  24.71156 129.1163
## 17.42857      170.12068 135.98740 204.2540 117.91833 222.3230
## 17.57143      175.76910 141.63581 209.9024 123.56674 227.9715
```

```r
# Extract point forecasts
forecast_values_ets <- final_ets_forecast$mean

# Create forecast dates from 16/06/2015 to 29/06/2015
forecast_dates_ets <- seq(as.Date("2015-06-16"), by = "days", length.out = 14)

# Create a data frame with dates and point forecasts
final_forecast_data_ets <- data.frame(date = forecast_dates_ets, forecast = forecast_values_ets)

# Print the final forecast data
print(final_forecast_data_ets)
```

```
##          date  forecast
## 1  2015-06-16 159.79583
## 2  2015-06-17 173.08458
## 3  2015-06-18 164.56924
## 4  2015-06-19 101.27350
## 5  2015-06-20  76.91392
## 6  2015-06-21 170.12068
## 7  2015-06-22 175.76910
## 8  2015-06-23 159.79583
## 9  2015-06-24 173.08458
## 10 2015-06-25 164.56924
## 11 2015-06-26 101.27350
## 12 2015-06-27  76.91392
## 13 2015-06-28 170.12068
## 14 2015-06-29 175.76910
```

Part 2 Conclusion:

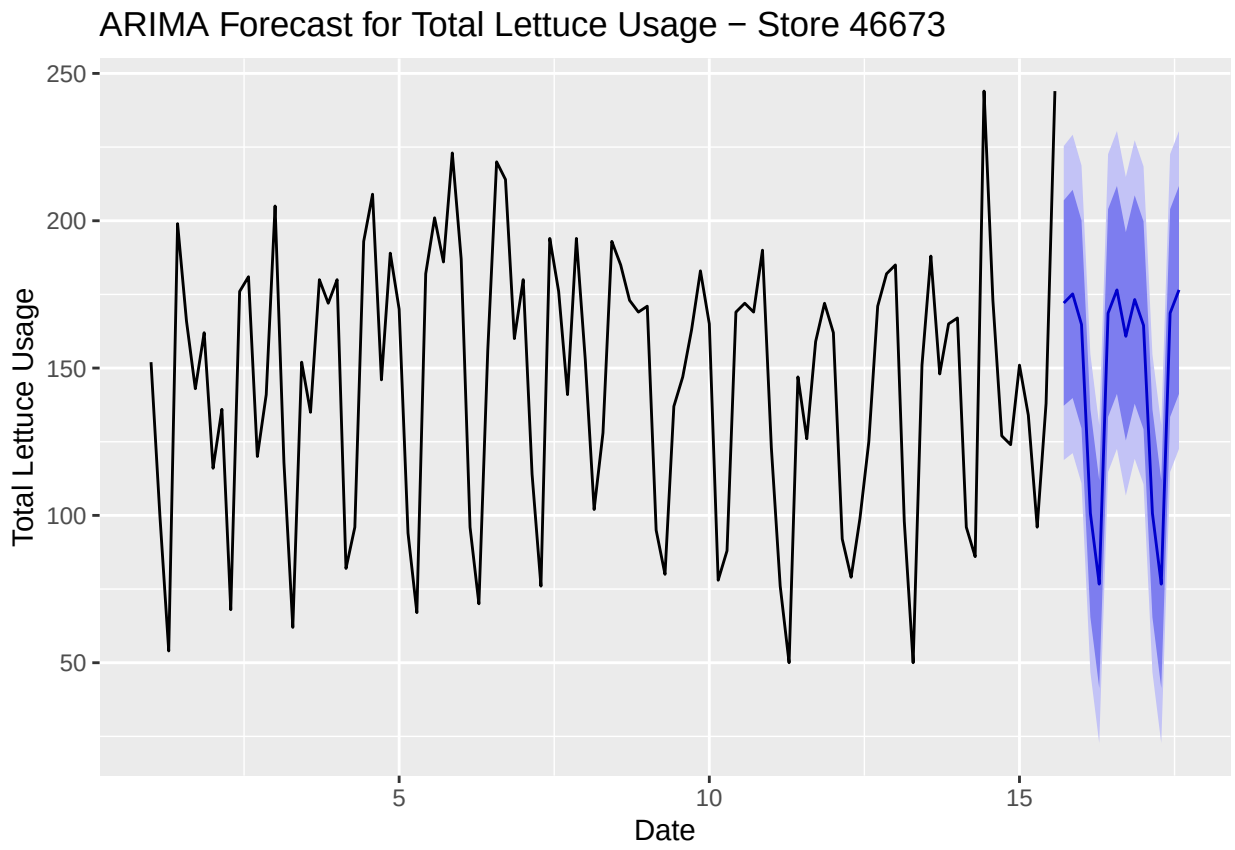
The resulting 'final_forecast_data_ets' contains the forecasted values and allows me to project future trends and seasonality based on the characteristics learned from the historical data.

Part 3 Comparison: Now, that we have both the Arima and the Holt winters model setup, we need to assess the forecasting performance of both the ARIMA and Holt-Winters models for a two-week timeframe.

To gain insights into the accuracy of each model's predictions, we initiated the examination by visualizing the forecasted values individually for ARIMA and Holt-Winters. This step allows us to observe the pattern and behavior of the forecasted series generated by each model.

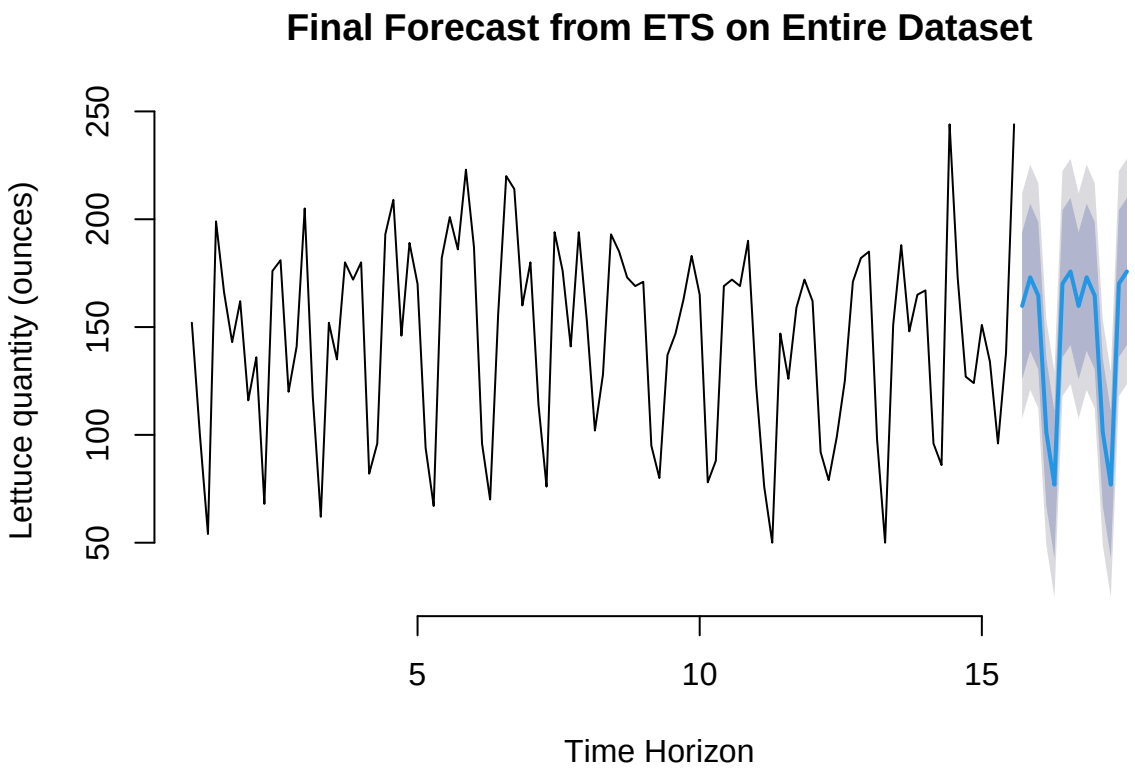
A) Plot the forecasts of the ARIMA Model:

```r
# Plot the forecast
autoplot(final_forecast_46673) +
  labs(title = "ARIMA Forecast for Total Lettuce Usage - Store 46673",
       x = "Date",
       y = "Total Lettuce Usage")
```



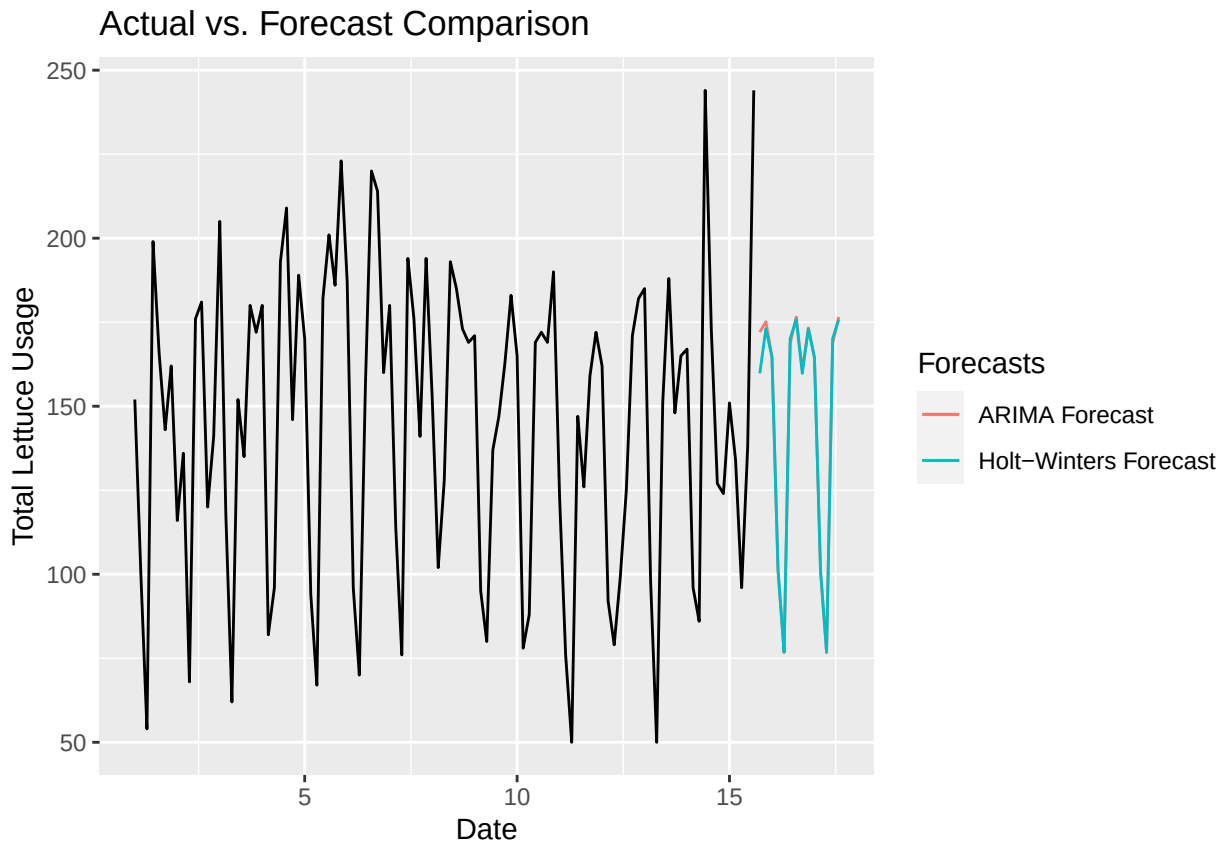
B) Plot the forecasts of the Holt Winter Model:

```r
# Plot the final forecast
plot(final_ets_forecast, main = "Final Forecast from ETS on Entire Dataset", xlab="Time Horizon", ylab=
```



C) ARIMA vs HoltWinters Model Forecasts:

```
# Plot actual data
autoplot(store_series_ts) +
  labs(title = "Actual vs. Forecast Comparison",
       x = "Date",
       y = "Total Lettuce Usage") +
  autolayer(final_forecast_46673$mean, series = "ARIMA Forecast") +
  autolayer(final_ets_forecast$mean, series = "Holt-Winters Forecast") +
  guides(color = guide_legend(title = "Forecasts"))
```



Analysis of the Accuracies of both the Models:

Following the visual analysis of the ARIMA and Holt-Winters forecasts, we proceeded to a quantitative assessment using the Root Mean Squared Error (RMSE). RMSE is a widely utilized metric for evaluating the accuracy of forecasting models, providing a comprehensive measure of the differences between predicted and observed values. By calculating the RMSE for both the ARIMA and Holt-Winters models, we aim to quantify the level of accuracy each model achieves in capturing the actual values. A lower RMSE signifies better predictive performance, indicating that the model's forecasts closely align with the observed data. This numerical comparison aids in selecting the model that exhibits superior accuracy and reliability in forecasting the daily demand for lettuce in our dataset over the specified two-week period.

```
#accuracy of arima
accuracy(arima_forecast_46673.m1, validation_set_46673)
```

```
##                      ME     RMSE      MAE       MPE     MAPE      MASE
## Training set -0.7949027 24.95910 18.28022 -2.980306 13.47389 0.6903407
## Test set     11.7965214 38.00071 26.92164  3.587282 18.18053 1.0166781
##                    ACF1 Theil's U
## Training set 0.005226471        NA
## Test set     0.101673824 0.5088318
```

```
#accuracy of HoltWinters
out_of_sample_errors_ets
```

```
##                      ME     RMSE      MAE       MPE     MAPE      MASE
## Training set -1.861661 23.22218 18.45667 -4.154372 14.30555 0.6970043
```

```
## Test set     22.863318 38.16146 29.06228 13.661495 19.14726 1.0975182
##                    ACF1 Theil's U
## Training set 0.05622874        NA
## Test set     0.04582855 0.4904983
```

```
#Preferred ARIMA
```

RMSE Values	ARIMA	Holt Winters
training set	24.95910	23.22218
test set	38.00071	38.16146

Key Insights:

1) The ANA model exhibits a lower RMSE on the training set, indicating that it fits the historical data more closely.

2) The ARIMA model performs better on the test set as it has a lower RMSE. This suggests that the ARIMA model's forecasting accuracy is better when applied to unseen data.

3) The visual inspection of forecasted values from both models reveals similar patterns. This observation aligns with the notion that, in some cases, different models may produce comparable forecasts even if their underlying structures are different.

To further analyse a better model, I calculated the AIC values of the models on which the ARIMA and Holtwinters were applied.

```
#ANA Model
AIC(store_46673_ets_training)
```

```
## [1] 897.1487
```

```
#ARIMA Model
AIC(arima_model_46673.m1)
```

```
## [1] 712.8527
```

The AIC values provide a quantitative measure of model fit, considering both the goodness of fit and model complexity. A lower AIC indicates a better-fitting model.

In this case, the ARIMA model has a lower AIC (712.8527) compared to the ANA model (897.1487), reinforcing the notion that the ARIMA model is preferred in terms of AIC.

While the ANA model shows better performance on the training set, the ARIMA model outperforms on the test set and has a lower AIC. The decision between the two models should consider both training and test set performance, and in this case, the ARIMA model appears to be more suitable for forecasting.

Final Forecasted Values:

Hence, the final predictions for the Store 46673 using the ARIMA model are:

```r
# Print the final forecast data
print(final_forecast_data_46673)
```

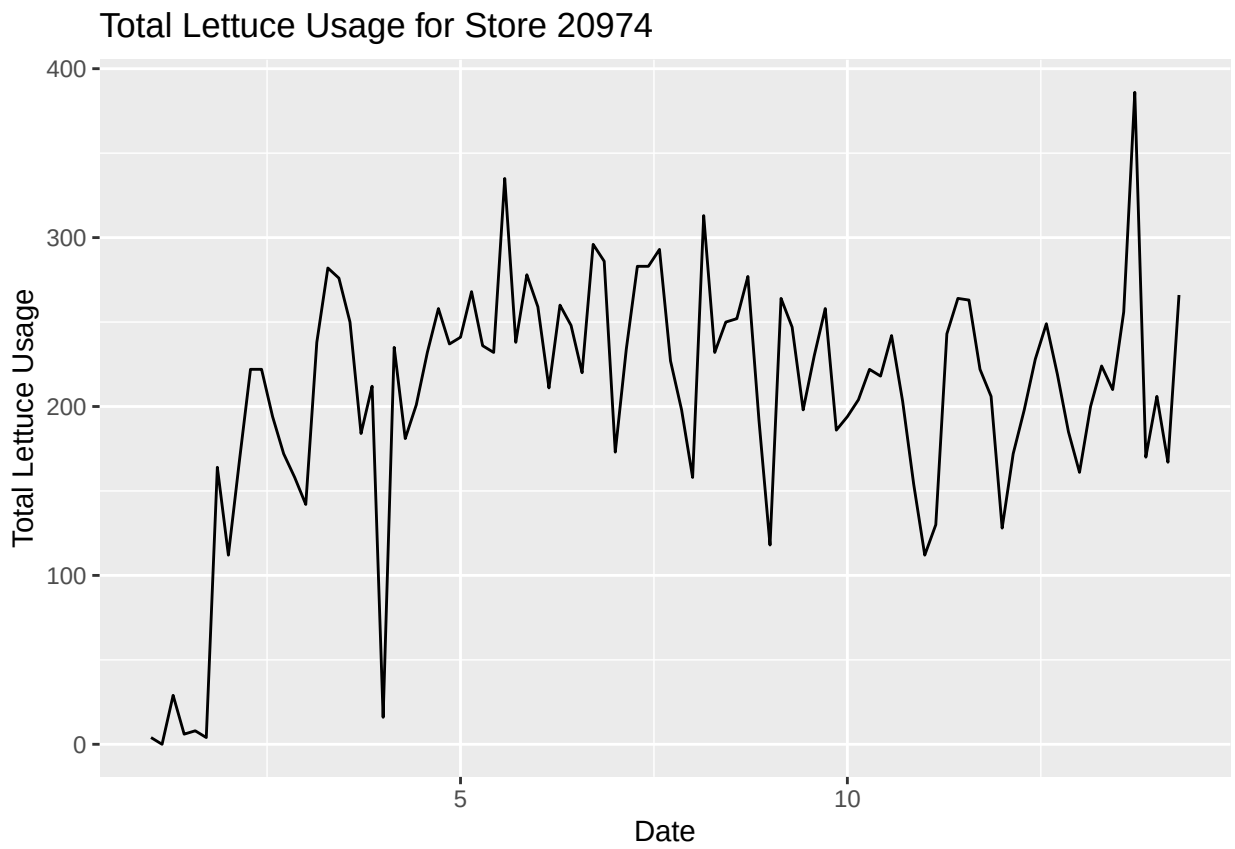
```
##          date  forecast
## 1  2015-06-16 172.06567
## 2  2015-06-17 175.15870
## 3  2015-06-18 164.80336
## 4  2015-06-19 100.78962
## 5  2015-06-20  76.67597
## 6  2015-06-21 168.66758
## 7  2015-06-22 176.52969
## 8  2015-06-23 160.78112
## 9  2015-06-24 173.27133
## 10 2015-06-25 164.48770
## 11 2015-06-26 100.73682
## 12 2015-06-27  76.66714
## 13 2015-06-28 168.66610
## 14 2015-06-29 176.52944
```

Store: 20974

In this analysis, we focused on a specific store, identified by the store number 20974. For our targeted store, we filtered the data to isolate lettuce-related transactions, creating a time series object to capture the daily total lettuce usage. The time series plot visualizes the fluctuations in lettuce consumption over time, providing insights into potential patterns or trends.

This approach allows us to gain a deeper understanding of the store's lettuce demand dynamics, a crucial factor for making informed inventory replenishment decisions.

```r
# Choose store number
store_number <- 20974

# Filter data for the chosen store
store_data_20974 <- total_lettuce_per_transaction %>% filter(StoreNumber == store_number)

# Create a zoo object
store_series <- zoo(store_data_20974$Total_Lettuce, as.Date(store_data_20974$date))

# Create a time series object
store_series_ts <- ts(store_data_20974$Total_Lettuce, frequency = 7)

# Plot the time series
autoplot(store_series_ts) +
  labs(title = paste("Total Lettuce Usage for Store", store_number),
       x = "Date",
       y = "Total Lettuce Usage")
```



We zoomed in on Store 20974, carefully dividing its lettuce usage data into two parts – an 80% chunk for training our forecasting models and a 20% slice to validate their accuracy. This split, respecting the chronological order of our data, sets the stage for effective model training and evaluation.

```
split_proportion <- 0.8

# Function to split data for a specific store with window adjustment
split_data <- function(store_data, split_proportion) {
  split_index <- round(length(store_data) * split_proportion)
  train_data <- window(store_data, end = time(store_data)[split_index])
  test_data <- window(store_data, start = time(store_data)[split_index + 1])
  return(list(train_data = train_data, test_data = test_data))
}

# Split data for each store with window adjustment
split_20974 <- split_data(store_series_ts, split_proportion)

# Training Set (80%)
training_set_20974_ts <- split_20974$train_data

# Validation Set (20%)
validation_set_20974 <- split_20974$test_data
```

Part 1) ARIMA Model: I performed stationarity tests using the Augmented Dickey-Fuller (ADF), Phillips-Perron (PP), and Kwiatkowski-Phillips-Schmidt-Shin (KPSS) tests. Additionally, ndiffs and nsdiffs

functions are used to determine the number of non-seasonal and seasonal differences needed to make the time series stationary, respectively.

```
# ARIMA identification and order determination
adf.test(training_set_20974_ts)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  training_set_20974_ts
## Dickey-Fuller = -3.887, Lag order = 4, p-value = 0.01953
## alternative hypothesis: stationary
```

```
pp.test(training_set_20974_ts)
```

```
## Warning in pp.test(training_set_20974_ts): p-value smaller than printed p-value
```

```
##
##  Phillips-Perron Unit Root Test
##
## data:  training_set_20974_ts
## Dickey-Fuller Z(alpha) = -28.711, Truncation lag parameter = 3, p-value
## = 0.01
## alternative hypothesis: stationary
```

```
kpss.test(training_set_20974_ts)
```

```
##
##  KPSS Test for Level Stationarity
##
## data:  training_set_20974_ts
## KPSS Level = 0.65978, Truncation lag parameter = 3, p-value = 0.0172
```

```
# seasonal differencing
ndiffs(training_set_20974_ts)
```

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

```
nsdiffs(training_set_20974_ts)
```

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

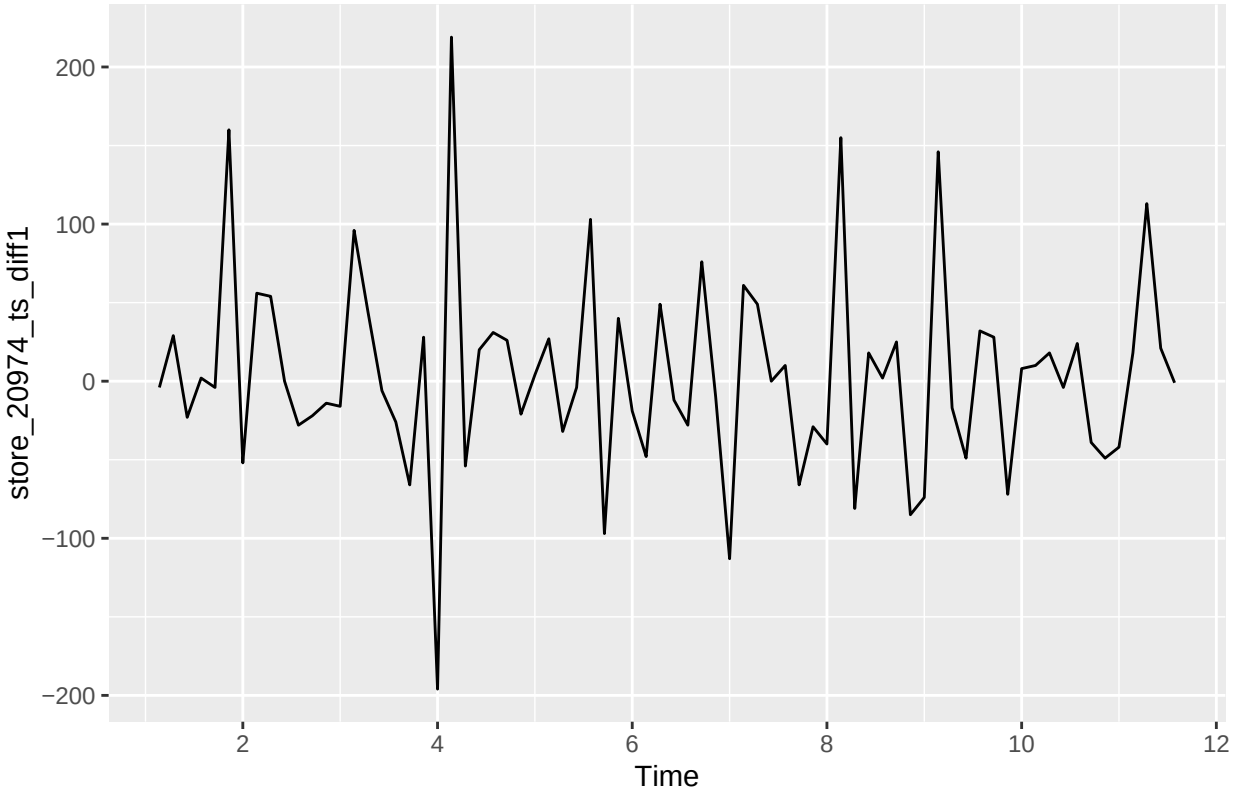
Key insights:

1. The ADF test suggests that the time series is stationary . The PP test also indicates stationarity as the p-value is below the significance level.
2. The KPSS test provides additional confirmation of stationarity, with a p-value below the significance level.
3. One non-seasonal difference is needed for stationarity.

In summary, based on these tests, the time series appears to be stationary after one non-seasonal difference.

Hence, we carry out differencing and plot our resulted data:

```
# ARIMA identification and order determination
store_20974_ts_diff1 <- diff(training_set_20974_ts, differences = 1)
autoplot(store_20974_ts_diff1)
```



Key insight: After differencing, data appears to be stationary now.

Now we recheck the required results through the same tests again:

```
#Now again check for all three tests
adf.test(store_20974_ts_diff1)
```

```
## Warning in adf.test(store_20974_ts_diff1): p-value smaller than printed p-value
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  store_20974_ts_diff1
## Dickey-Fuller = -6.1354, Lag order = 4, p-value = 0.01
## alternative hypothesis: stationary
```

```
pp.test(store_20974_ts_diff1)
```

```
## Warning in pp.test(store_20974_ts_diff1): p-value smaller than printed p-value
```

```
##
##  Phillips-Perron Unit Root Test
##
## data:  store_20974_ts_diff1
## Dickey-Fuller Z(alpha) = -90.093, Truncation lag parameter = 3, p-value
## = 0.01
## alternative hypothesis: stationary
```

```
kpss.test(store_20974_ts_diff1)
```

```
## Warning in kpss.test(store_20974_ts_diff1): p-value greater than printed
## p-value
```

```
##
##  KPSS Test for Level Stationarity
##
## data:  store_20974_ts_diff1
## KPSS Level = 0.0972, Truncation lag parameter = 3, p-value = 0.1
```

```
# seasonal differencing
ndiffs(store_20974_ts_diff1)
```

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

```
nsdiffs(store_20974_ts_diff1)
```

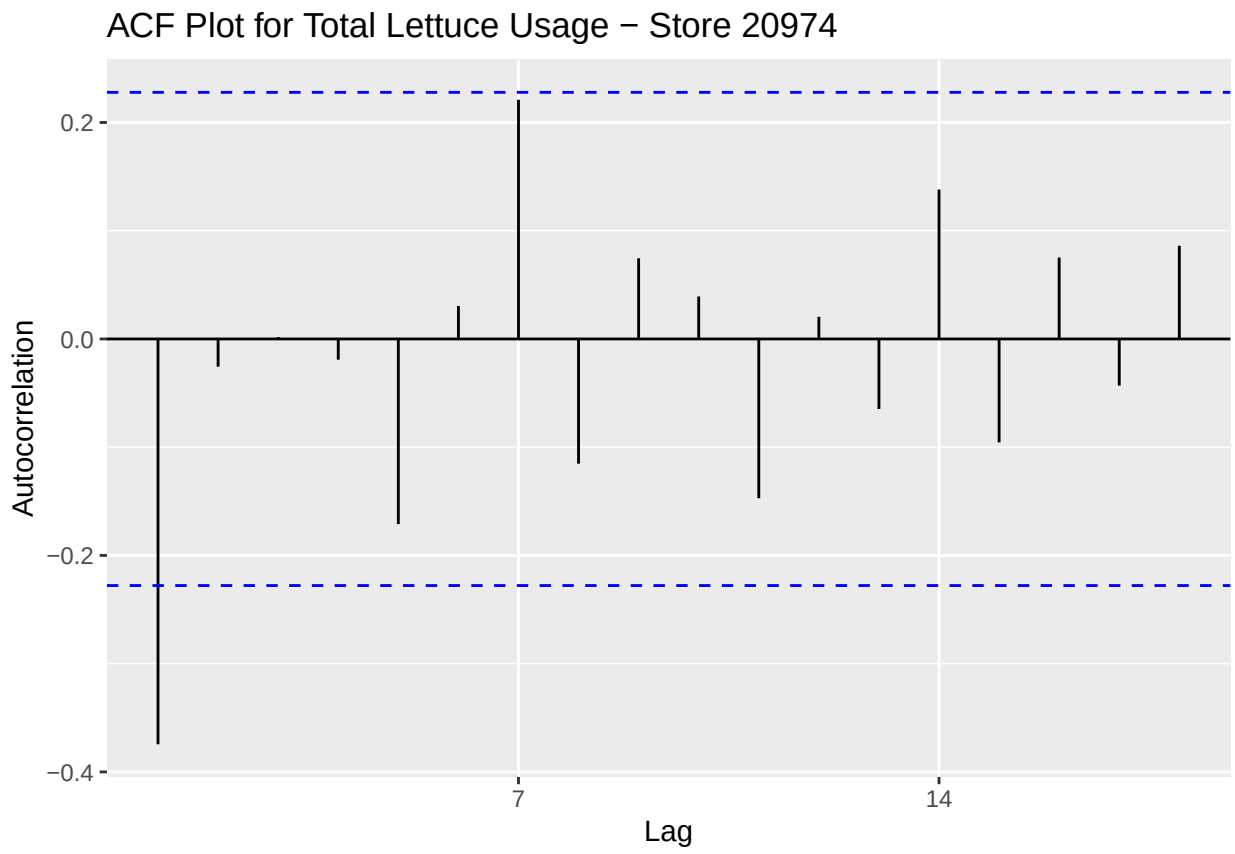
```
## [1] 0
```

The differenced time series, obtained through seasonal differencing, exhibits strong evidence of stationarity based on Augmented Dickey-Fuller and Phillips-Perron tests, supporting its suitability for further modeling and forecasting.

To further solidfy the selection of appropriate ARIMA parameters for our time series, we turn to the analysis of the autocorrelation function (ACF) and partial autocorrelation function (PACF). The visual examination of ACF and PACF plots aids in determining the optimal order for the ARIMA model, facilitating a more accurate and effective forecasting approach.

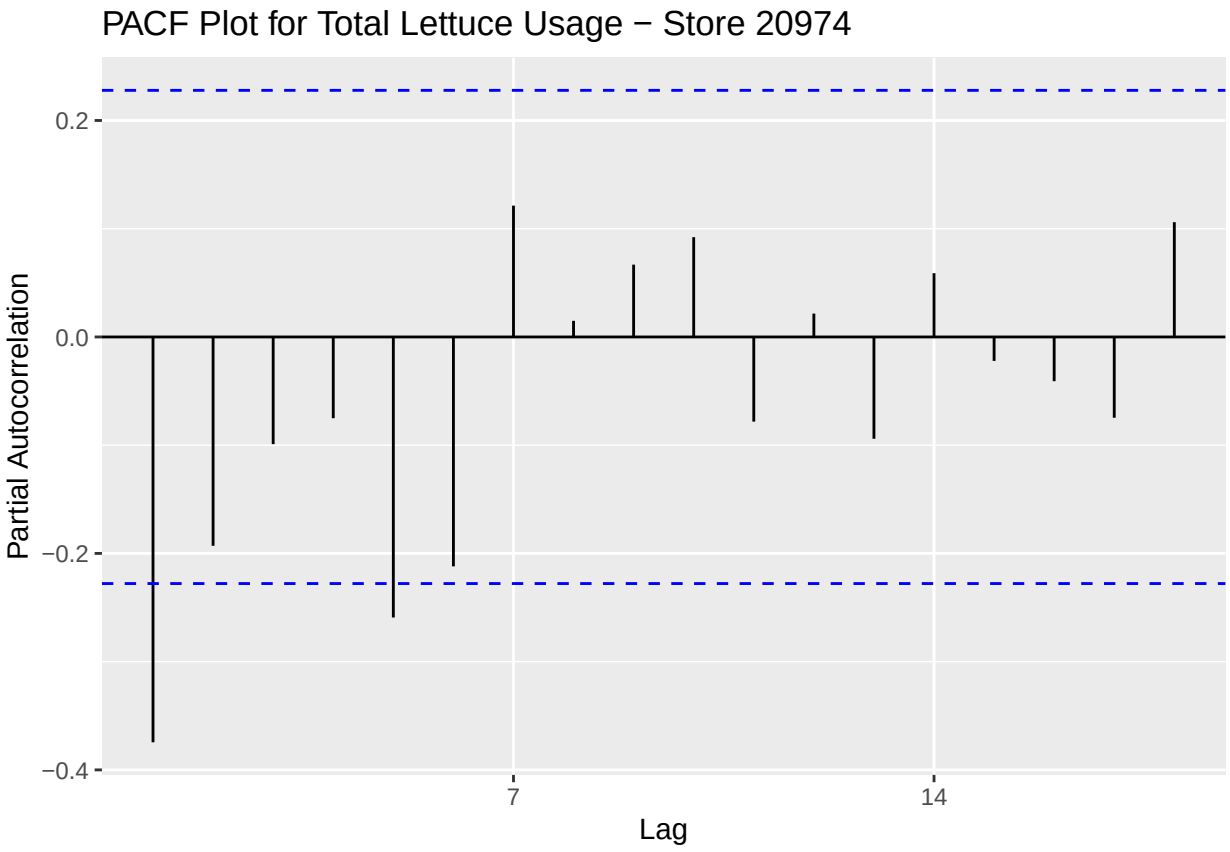
ACF Plot:

```
# Determine p and q
ggAcf(store_20974_ts_diff1) +
  labs(title = "ACF Plot for Total Lettuce Usage - Store 20974",
       x = "Lag",
       y = "Autocorrelation")
```



PACF plot:

```
ggPacf(store_20974_ts_diff1) +
  labs(title = "PACF Plot for Total Lettuce Usage - Store 20974",
       x = "Lag",
       y = "Partial Autocorrelation")
```



Model Selection:

The auto.arima function was used to explore various combinations of differencing and seasonal parameters, considering the Akaike Information Criterion (AIC) as the selection criterion. Upon analysing the AIC values, three models with the lowest AICs were chosen. Subsequently, we proceeded to fit these selected models on our training data to further evaluate their forecasting capabilities.

(The code line for generating the auto arima models 'auto_arima_result_20974' is commented because of the long series of output. However, it stays uncommented in the RMD file)

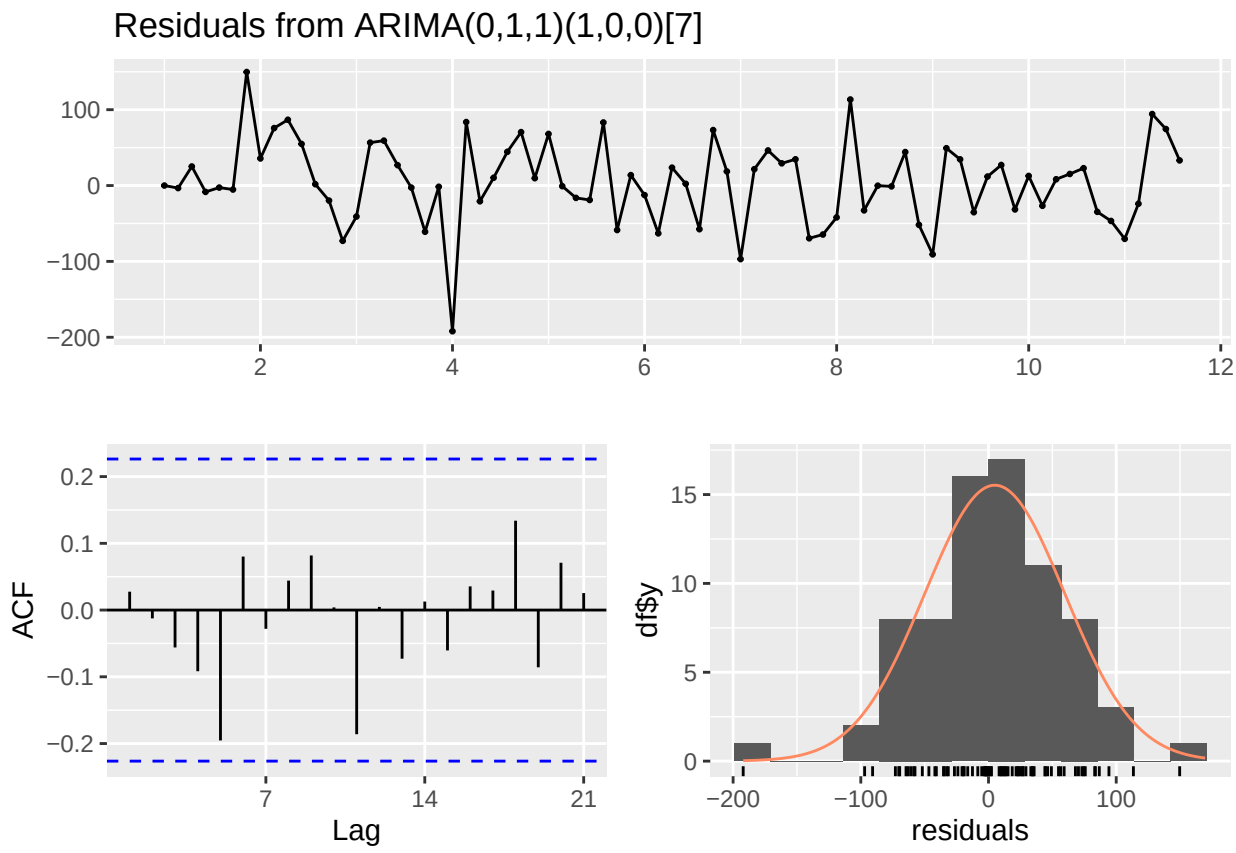
```
# Automatic ARIMA model selection
#auto_arima_result_20974 <- auto.arima(training_set_20974_ts,
#D = 0, d = 1, ic = 'aicc', trace = TRUE, stepwise = FALSE, approximation = FALSE)

#auto_arima_result_20974

#Best Models selection:
#ARIMA(0,1,1)(1,0,0)[7]: 810.7419
#ARIMA(0,1,1)(0,0,1)[7]: 811.4762
#ARIMA(0,1,1)(1,0,0)[7] with drift: 812.01


# Fit the ARIMA model
arima_model_20974.m1 <- Arima(training_set_20974_ts, order = c(0, 1, 1),
seasonal = list(order = c(1, 0, 0), period = 7), include.drift = FALSE)

arima_model_20974.m2 <- Arima(training_set_20974_ts, order = c(0, 1, 1),
```

```
seasonal = list(order = c(0, 0, 1), period = 7), include.drift = FALSE)

arima_model_20974.m3 <- Arima(training_set_20974_ts, order = c(0, 1, 1),
seasonal = list(order = c(1, 0, 0), period = 7), include.drift = TRUE)
```

Residuals analysis:

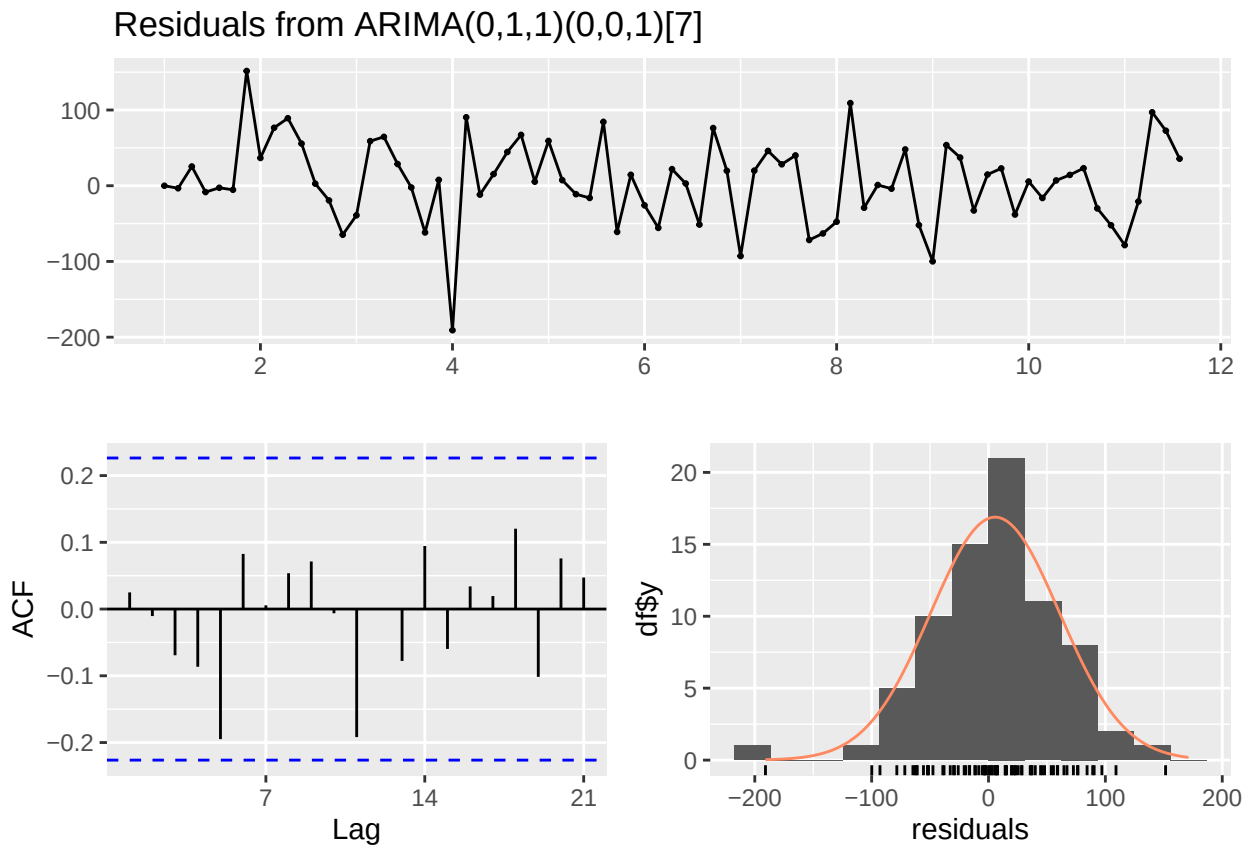
To make sure that the models we selected effectively represent the temporal trends, we used residual analysis. For residuals, R's checkresiduals() function offers an extensive collection of statistical tests and diagnostic charts. In order to confirm the accuracy of our projections and to make appropriate judgments, this phase is essential.

```
# Residual analysis
checkresiduals(arima_model_20974.m1)
```



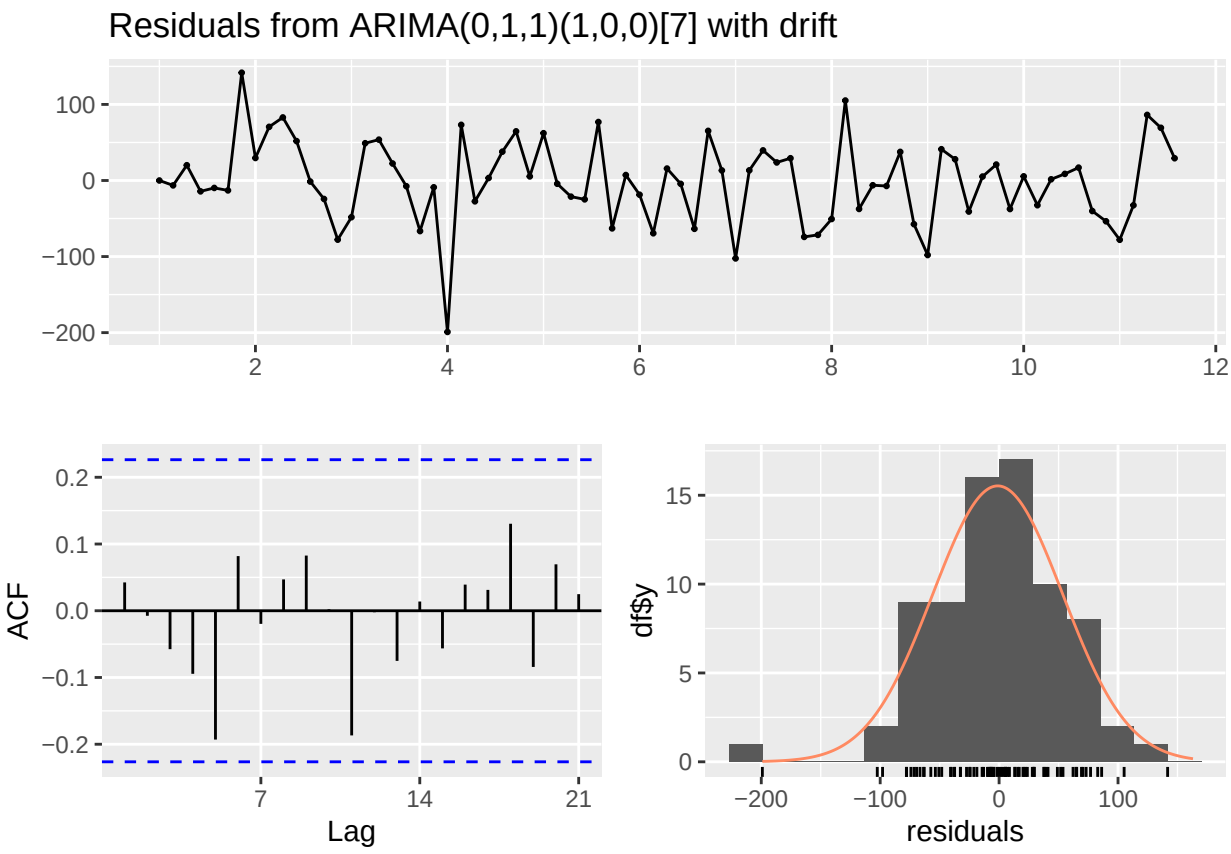

```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(0,1,1)(1,0,0)[7]
## Q* = 9.1476, df = 12, p-value = 0.6903
##
## Model df: 2.   Total lags used: 14
```

```
checkresiduals(arima_model_20974.m2)
```




```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(0,1,1)(0,0,1)[7]
## Q* = 10.183, df = 12, p-value = 0.5999
##
## Model df: 2.   Total lags used: 14
```

```
checkresiduals(arima_model_20974.m3)
```




```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(0,1,1)(1,0,0)[7] with drift
## Q* = 9.2851, df = 12, p-value = 0.6784
##
## Model df: 2.   Total lags used: 14
```

Forecasting and Accuracy:

Considering the fitted ARIMA models, we produce forecasts for the validation set in this stage. The validation set is a certain fraction of our time series data that was not used for model training. The 'accuracy' function, which offers a comprehensive set of parameters to evaluate the model's performance against the real data, is then used to determine how accurate these forecasts are. This analysis helps determine which ARIMA model is most likely to accurately estimate lettuce demand at Store 20974.

```
# Forecast using the fitted model
arima_forecast_20974.m1 <- forecast(arima_model_20974.m1, h = length(validation_set_20974))
arima_forecast_20974.m2 <- forecast(arima_model_20974.m2, h = length(validation_set_20974))
arima_forecast_20974.m3 <- forecast(arima_model_20974.m3, h = length(validation_set_20974))


# Model evaluation
accuracy(arima_forecast_20974.m1, validation_set_20974)
```

```
##                      ME      RMSE       MAE       MPE      MAPE      MASE
```

```
## Training set   4.913088 54.74107 41.66443      -Inf      Inf 0.7072345
## Test set     -24.682418 53.87050 42.62524 -16.64612 21.66534 0.7235437
##                     ACF1 Theil's U
## Training set  0.02755342        NA
## Test set     -0.03988790 0.8827914
```

```
accuracy(arima_forecast_20974.m2, validation_set_20974)
```

```
##                      ME     RMSE      MAE       MPE     MAPE      MASE
## Training set   5.645958 55.06105 41.89174      -Inf      Inf 0.7110929
## Test set     -24.924964 55.51908 44.11241 -17.00851 22.47915 0.7487878
##                    ACF1 Theil's U
## Training set 0.02496576        NA
## Test set     0.01267890 0.9220209
```

```
accuracy(arima_forecast_20974.m3, validation_set_20974)
```

```
##                      ME     RMSE      MAE       MPE     MAPE      MASE
## Training set  -1.148739 54.37960 41.36197      -Inf      Inf 0.7021003
## Test set     -56.648639 74.43427 66.42792 -31.93169 34.46519 1.1275833
##                     ACF1 Theil's U
## Training set  0.04245734        NA
## Test set     -0.01981440  1.282767
```

```
#best rmse value is for arima_forecast_20974.m1 model
```

Key Insights:

1. With the lowest RMSE on the test sets, Model 1 performs better than the others according to the evaluation metrics, indicating that it is capable of forecasting lettuce demand at Store 20974 with greater accuracy.
2. Comparing this to models m2 and m3, a smaller RMSE demonstrates better performance in minimizing the prediction error.
3. The smaller RMSE values indicate that Model 1's forecasts align more closely with the actual values, making it the preferred choice for lettuce demand prediction in this context.

Fitting the best ARIMA Model on the entire data set:

The selected ARIMA(0,1,1)(1,0,0)[7] model, which has been determined to be the best-performing model on the training and validation sets, is being fitted to the whole dataset for Store 20974 in this stage of development. Establishing a definitive forecast for the subsequent 14 days is the ultimate objective.

```
#now fit the first best model on our entire dataset
arima_model_20974_final <- Arima(store_series_ts, order = c(0, 1, 1),
seasonal = list(order = c(1, 0, 0), period = 7), include.drift = FALSE)

# Final forecast
final_forecast_20974 <- forecast(arima_model_20974_final, h = 14)

# Extract point forecasts
forecast_values_20974 <- final_forecast_20974$mean
```

```r
# Create forecast dates from 16/06/2015 to 29/06/2015
forecast_dates_20974 <- seq(as.Date("2015-06-16"), by = "days", length.out = 14)

# Create a data frame with dates and point forecasts
final_forecast_data_20974 <- data.frame(date = forecast_dates_20974, forecast = forecast_values_20974)

# Print the final forecast data
print(final_forecast_data_20974)
```

```
##          date forecast
## 1  2015-06-16 230.0867
## 2  2015-06-17 247.3005
## 3  2015-06-18 295.9482
## 4  2015-06-19 215.1182
## 5  2015-06-20 228.5899
## 6  2015-06-21 213.9956
## 7  2015-06-22 251.0426
## 8  2015-06-23 237.6034
## 9  2015-06-24 244.0450
## 10 2015-06-25 262.2496
## 11 2015-06-26 232.0020
## 12 2015-06-27 237.0433
## 13 2015-06-28 231.5819
## 14 2015-06-29 245.4454
```

Part 1 Conclusion:

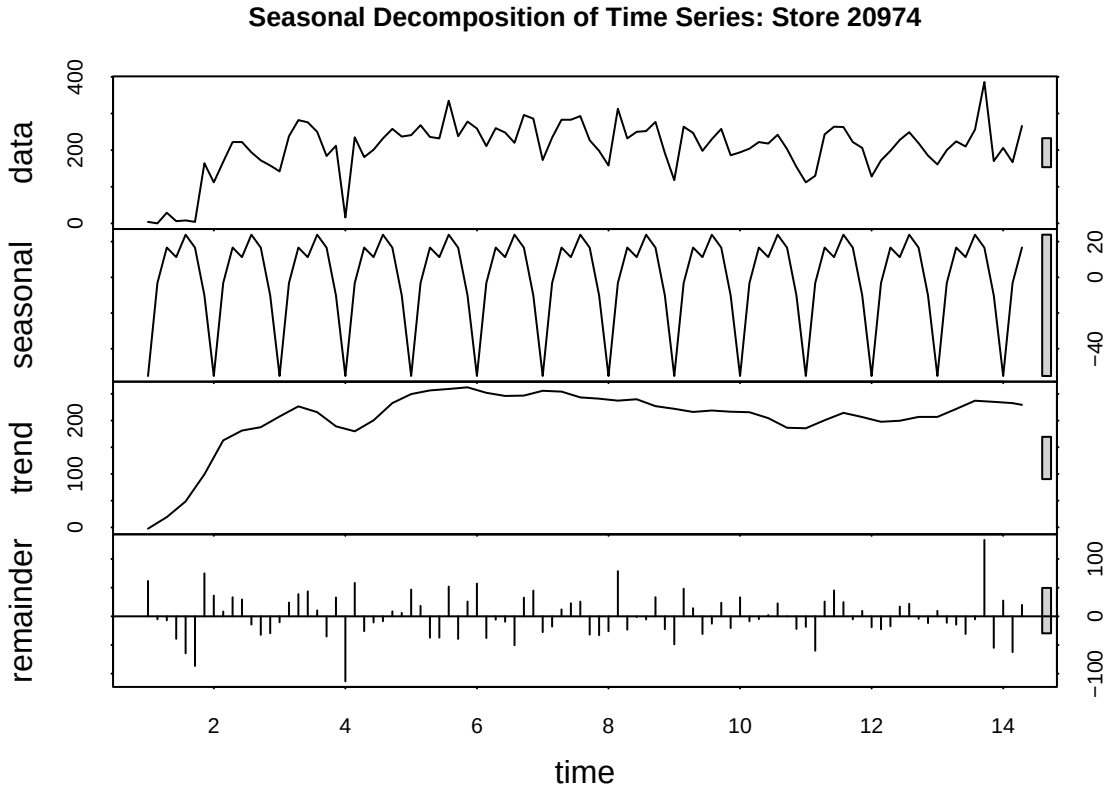
In summary, the model has been validated on historical data, and its accuracy assessed through metrics such as RMSE. We built a forecast for the next 14 days using this ARIMA model, which offers insightful information for inventory management.

Part 2) Holt Winters Model: Moving forward, we will extend our analysis by investigating the Holt-Winters model in order to determine the best method of forecasting the demand for lettuce in our dataset.

Seasonal Decomposition:

Decomposing a time series facilitates the process of recognizing and illustrating the underlying patterns in the data, including the general trend and recurring seasonal patterns. Determining the total variability in the time series is made easier by having a clear understanding of each component's contribution.

```r
plot(stl(store_series_ts, s.window = "period"), main="Seasonal Decomposition of Time Series: Store 2097
```



Key Insights:

1) The trend shows initial increase in the beginning, but is more or less stable for quite some time later. There is no linear trend for sure.

2) An additive seasonal component seems to take place in the time-series and will be needed to the upcoming forecast model.

Model selection:

I performed ETS (Exponential Smoothing State Space) forecasting using the training set for Store 20974. First, I used the ets() function without specifying a model (store_20974_ets_training2) to identify the best ETS model for the given data. I got (A, A, N) model as the best model in this case. However, its very evident that data follows a non-linear trend. Hence, In the context of our model analysis, (A, N, A) proves to be the most accurate in this case. I then used this model (store_20974_ets_training) on the training dataset for further analysis.

```
# ETS forecast based on the training set
store_20974_ets_training2 <- ets(training_set_20974_ts, model='ZZZ')

store_20974_ets_training <- ets(training_set_20974_ts, model='ANA', ic = 'aicc')
store_20974_ets_training
```

```
## ETS(A,N,A)
##
## Call:
```

```
##  ets(y = training_set_20974_ts, model = "ANA", ic = "aicc")
##
##   Smoothing parameters:
##     alpha = 0.5499
##     gamma = 1e-04
##
##   Initial states:
##     l = 113.4594
##     s = -5.8202 5.8384 22.1644 13.9221 19.3568 6.9014
##            -62.3629
##
##   sigma:  53.0905
##
##      AIC     AICc      BIC
## 930.0238 933.4613 953.1987
```

In-sample estimation error:

Here we are evaluating the in-sample estimation error of the ETS (Error, Trend, Seasonal) model using the accuracy() function, which provides metrics like Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and others.

```
# In-sample estimation error for ETS
in_sample_errors_ets <- accuracy(store_20974_ets_training)
in_sample_errors_ets
```

```
##                     ME     RMSE      MAE  MPE MAPE      MASE       ACF1
## Training set 2.890488 49.80332 39.14105 -Inf  Inf 0.6644013 0.01092746
```

In conclusion, these metrics provide insights into the performance of the ETS model on the training data.

Out-of-sample evaluation:

I am using the trained ETS model (store_20974_ets_training) to generate forecasts for the length of the validation set. Furthermore, I evaluate the accuracy of the ETS model's forecasts against the actual values in the validation set. This helps in understanding how well the ETS model performs on unseen data by comparing its forecasts with the actual values in the validation set, using a range of accuracy metrics.

```
#Out-of-sample evaluation for ETS
store_20974_ets_forecast <- forecast.ets(store_20974_ets_training, h = length(validation_set_20974))

# Forecasting erros - ETS MODEL
out_of_sample_errors_ets <- accuracy(store_20974_ets_forecast, validation_set_20974)
out_of_sample_errors_ets
```

```
##                      ME     RMSE      MAE       MPE     MAPE      MASE
## Training set   2.890488 49.80332 39.14105      -Inf      Inf 0.6644013
## Test set     -17.469049 50.26356 38.33921 -12.13132 18.57596 0.6507905
##                     ACF1 Theil's U
## Training set  0.01092746        NA
## Test set     -0.17222234 0.8623348
```

Building the ANA model on the entire data set:

The trained model is now employed to forecast the future values for the entire time series (store_series_ts). The model is trained on the entire dataset, and then we generate a forecast for the next 14 time points (h = 14).

```
final_ets_model <- ets(store_series_ts, model='ANA', ic = 'aicc')
final_ets_forecast <- forecast(final_ets_model, h = 14)

# Extract point forecasts
forecast_values_ets <- final_ets_forecast$mean

# Create forecast dates from 16/06/2015 to 29/06/2015
forecast_dates_ets <- seq(as.Date("2015-06-16"), by = "days", length.out = 14)

# Create a data frame with dates and point forecasts
final_forecast_data_ets <- data.frame(date = forecast_dates_ets, forecast = forecast_values_ets)

# Print the final forecast data
print(final_forecast_data_ets)
```

```
##          date forecast
## 1  2015-06-16 239.2185
## 2  2015-06-17 254.3795
## 3  2015-06-18 248.0047
## 4  2015-06-19 219.4262
## 5  2015-06-20 171.2160
## 6  2015-06-21 229.1470
## 7  2015-06-22 245.2863
## 8  2015-06-23 239.2185
## 9  2015-06-24 254.3795
## 10 2015-06-25 248.0047
## 11 2015-06-26 219.4262
## 12 2015-06-27 171.2160
## 13 2015-06-28 229.1470
## 14 2015-06-29 245.2863
```

Part 2 Conclusion:

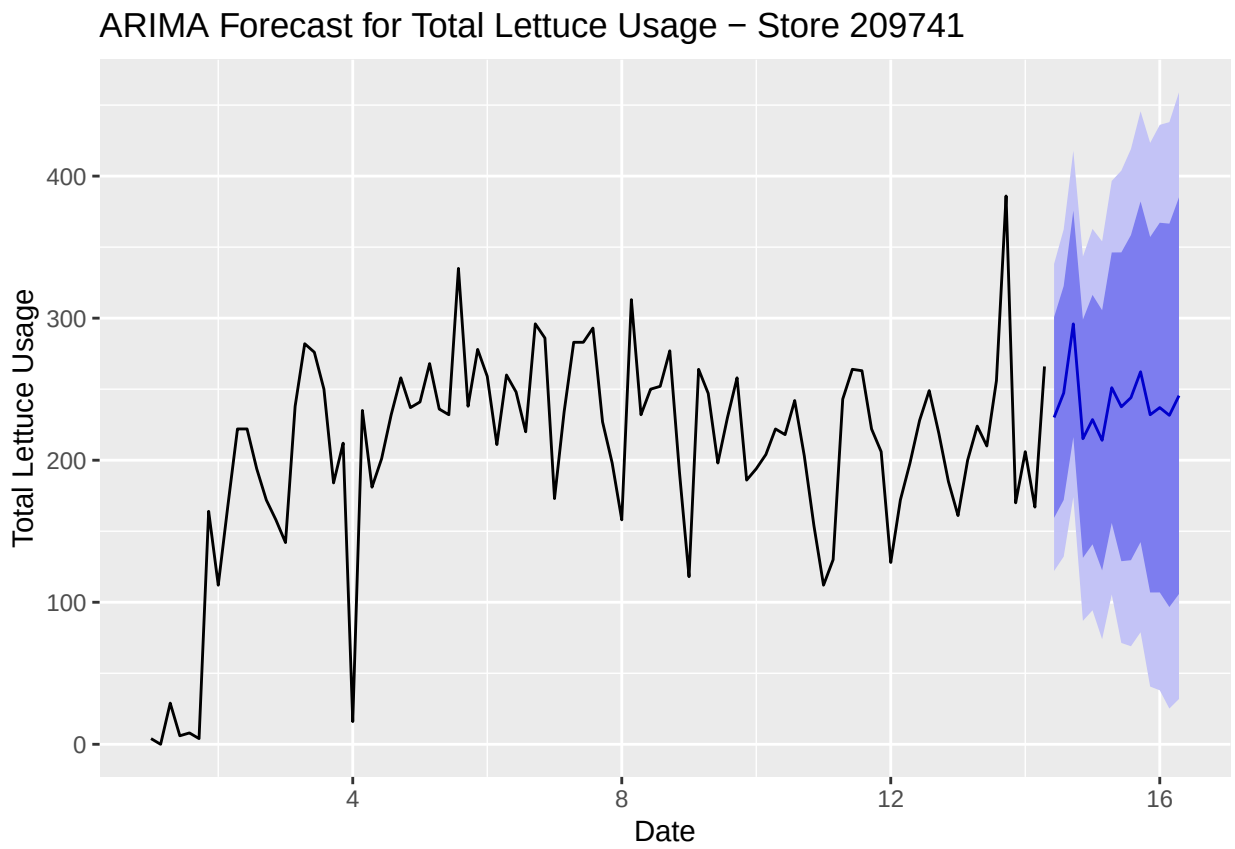
The resulting 'final_ets_forecast' contains the forecasted values and allows me to project future trends and seasonality based on the characteristics learned from the historical data.

Part 3 Comparison: Now, that we have both the Arima and the Holt winters model setup, we need to assess the forecasting performance of both the ARIMA and Holt-Winters models for a two-week timeframe.

To gain insights into the accuracy of each model's predictions, we initiated the examination by visualizing the forecasted values individually for ARIMA and Holt-Winters. This step allows us to observe the pattern and behavior of the forecasted series generated by each model.

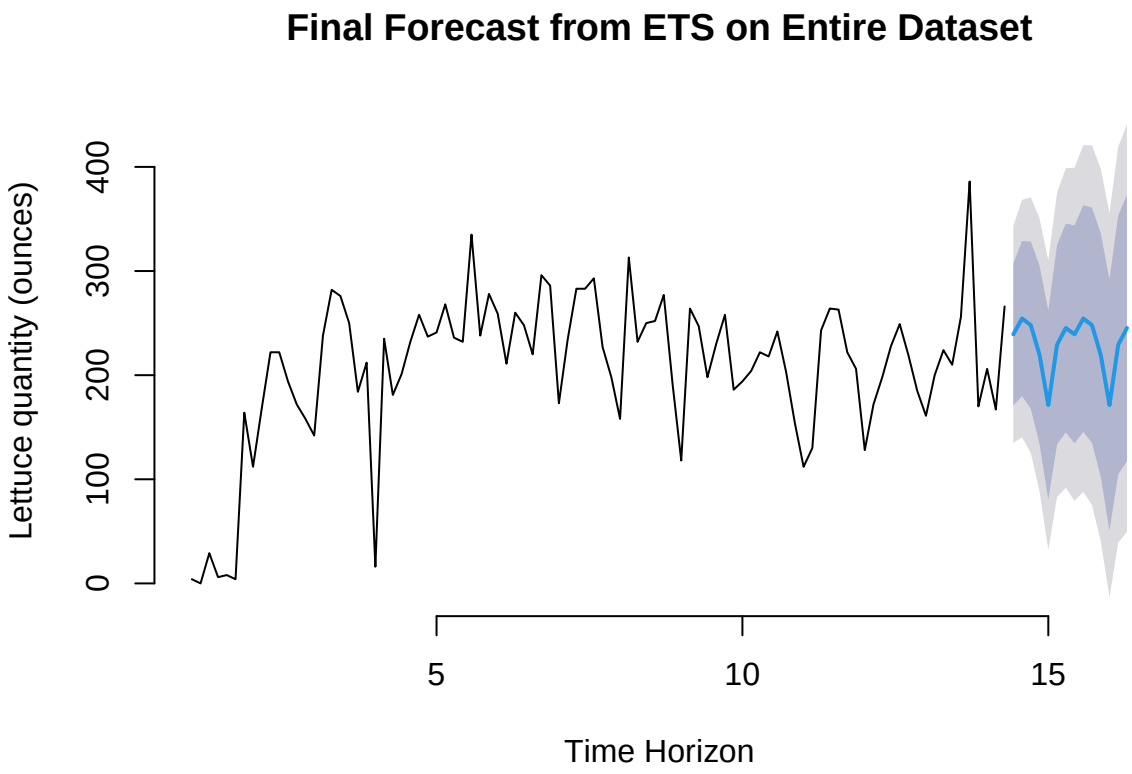
A) Plot the forecasts of the ARIMA Model:

```
# Plot the forecast
autoplot(final_forecast_20974) +
  labs(title = "ARIMA Forecast for Total Lettuce Usage - Store 209741",
       x = "Date",
       y = "Total Lettuce Usage")
```



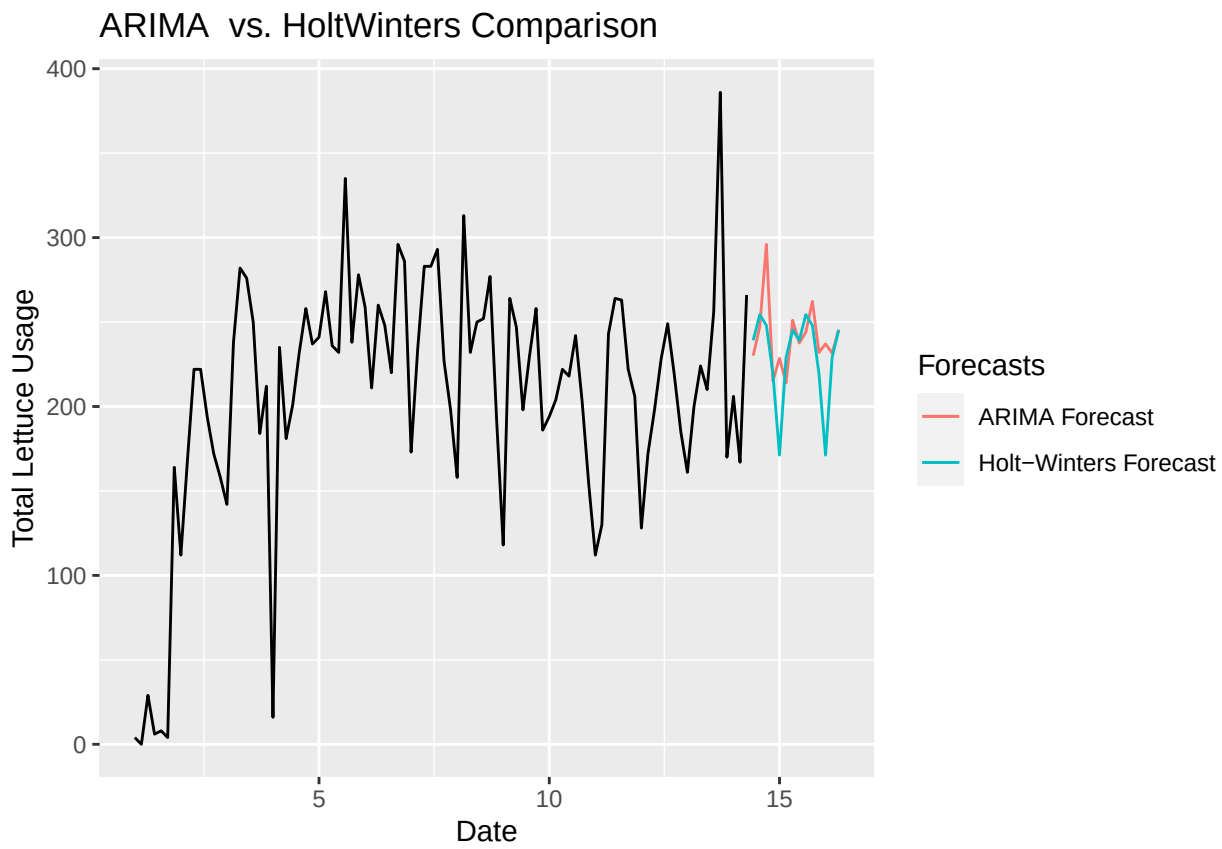
B) Plot the forecasts of the Holt Winter Model:

```
# Plot the final forecast
plot(final_ets_forecast, main = "Final Forecast from ETS on Entire Dataset", xlab="Time Horizon", ylab=
```



C) ARIMA vs HoltWinters Model Forecasts:

```
# Plot actual data
autoplot(store_series_ts) +
  labs(title = "ARIMA  vs. HoltWinters Comparison",
       x = "Date",
       y = "Total Lettuce Usage") +
  autolayer(final_forecast_20974$mean, series = "ARIMA Forecast") +
  autolayer(final_ets_forecast$mean, series = "Holt-Winters Forecast") +
  guides(color = guide_legend(title = "Forecasts"))
```



Analysis of the Accuracies of both the Models:

Following the visual analysis of the ARIMA and Holt-Winters forecasts, we proceeded to a quantitative assessment using the Root Mean Squared Error (RMSE). RMSE is a widely utilized metric for evaluating the accuracy of forecasting models, providing a comprehensive measure of the differences between predicted and observed values. By calculating the RMSE for both the ARIMA and Holt-Winters models, we aim to quantify the level of accuracy each model achieves in capturing the actual values. A lower RMSE signifies better predictive performance, indicating that the model's forecasts closely align with the observed data. This numerical comparison aids in selecting the model that exhibits superior accuracy and reliability in forecasting the daily demand for lettuce in our dataset over the specified two-week period.

```
#ARIMA Model
accuracy(arima_forecast_20974.m1, validation_set_20974)
```

```
##                      ME     RMSE      MAE       MPE     MAPE      MASE
## Training set   4.913088 54.74107 41.66443      -Inf      Inf 0.7072345
## Test set     -24.682418 53.87050 42.62524 -16.64612 21.66534 0.7235437
##                    ACF1 Theil's U
## Training set  0.02755342        NA
## Test set     -0.03988790 0.8827914
```

```
#ETS ANA Model
out_of_sample_errors_ets
```

```
##                      ME     RMSE      MAE       MPE     MAPE      MASE
## Training set   2.890488 49.80332 39.14105      -Inf      Inf 0.6644013
```

```
## Test set     -17.469049 50.26356 38.33921 -12.13132 18.57596 0.6507905
##                     ACF1 Theil's U
## Training set  0.01092746        NA
## Test set     -0.17222234 0.8623348
```

RMSE Values	ARIMA	HoltWinters
Training Set	54.74107	49.80332
Test set	53.87050	50.26356

Key Insights:

ETS model, the ANAModel, has lower RMSE values for both the training and test sets compared to the ARIMA model. This suggests that the ETS model provides a better fit to the data in terms of forecasting accuracy. The lower RMSE values indicate that the ETS model's predictions are closer to the actual values, both in the training and test phases.

In summary, based on the RMSE values, the ETS model outperforms the ARIMA model in this specific analysis.

Final Forecasted Values:

Hence, the final predictions for the Store 20974 using the ETS - ANA model are:

```
final_forecast_data_ets
```

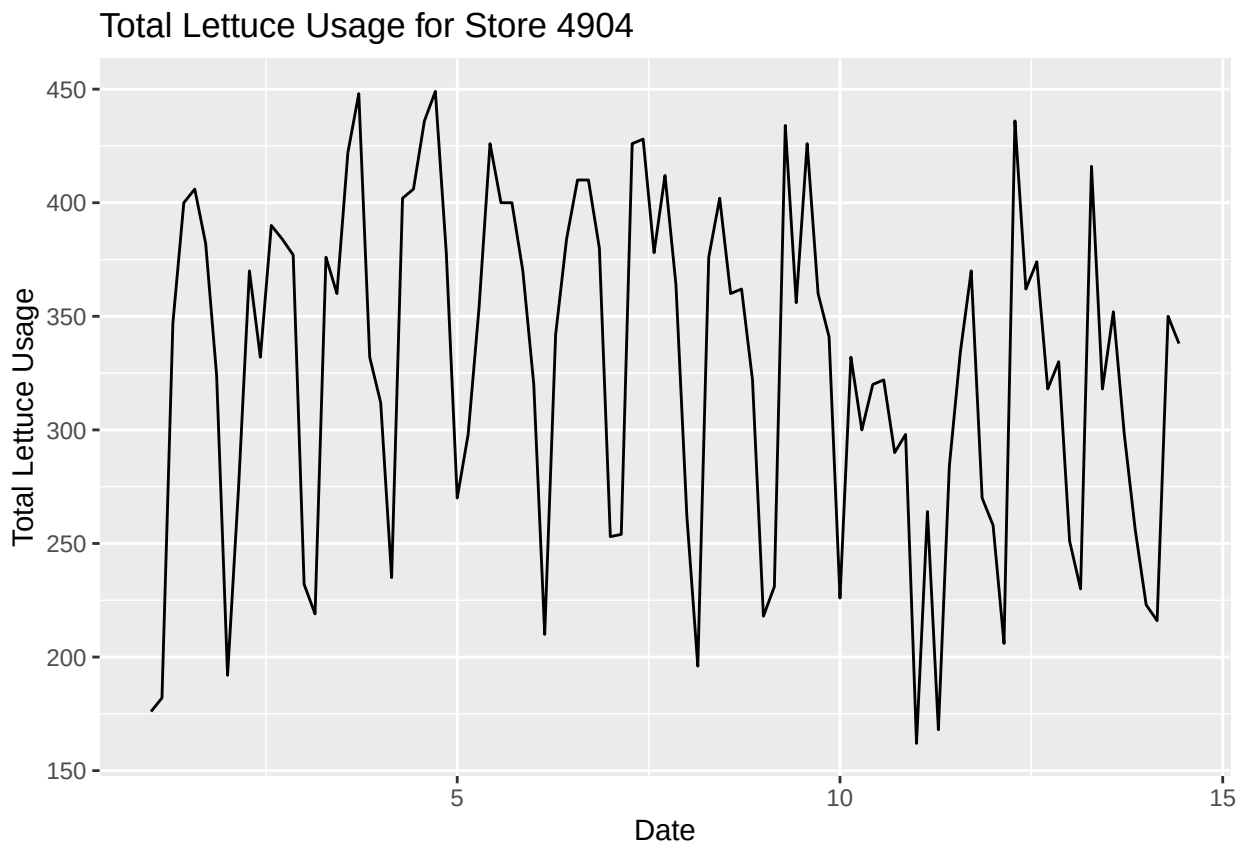
```
##          date forecast
## 1  2015-06-16 239.2185
## 2  2015-06-17 254.3795
## 3  2015-06-18 248.0047
## 4  2015-06-19 219.4262
## 5  2015-06-20 171.2160
## 6  2015-06-21 229.1470
## 7  2015-06-22 245.2863
## 8  2015-06-23 239.2185
## 9  2015-06-24 254.3795
## 10 2015-06-25 248.0047
## 11 2015-06-26 219.4262
## 12 2015-06-27 171.2160
## 13 2015-06-28 229.1470
## 14 2015-06-29 245.2863
```

Store: 4904

In this analysis, we focused on a specific store, identified by the store number 4904. For our targeted store, we filtered the data to isolate lettuce-related transactions, creating a time series object to capture the daily total lettuce usage. The time series plot visualizes the fluctuations in lettuce consumption over time, providing insights into potential patterns or trends.

This approach allows us to gain a deeper understanding of the store's lettuce demand dynamics, a crucial factor for making informed inventory replenishment decisions.

```r
# Choose store number
store_number <- 4904

# Filter data for the chosen store
store_data_4904 <- total_lettuce_per_transaction %>% filter(StoreNumber == store_number)

# Create a zoo object
store_series <- zoo(store_data_4904$Total_Lettuce, as.Date(store_data_4904$date))

# Create a time series object
store_series_ts <- ts(store_data_4904$Total_Lettuce, frequency = 7)

# Plot the time series
autoplot(store_series_ts) +
  labs(title = paste("Total Lettuce Usage for Store", store_number),
       x = "Date",
       y = "Total Lettuce Usage")
```



We zoomed in on Store 4904, carefully dividing its lettuce usage data into two parts – an 80% chunk for training our forecasting models and a 20% slice to validate their accuracy. This split, respecting the chronological order of our data, sets the stage for effective model training and evaluation.

```r
# Split data for each store with window adjustment
split_4904 <- split_data(store_series_ts, split_proportion)

# Training Set (80%)
```

```r
training_set_4904_ts <- split_4904$train_data

# Validation Set (20%)
validation_set_4904 <- split_4904$test_data
```

Part 1) ARIMA Model: I performed stationarity tests using the Augmented Dickey-Fuller (ADF), Phillips-Perron (PP), and Kwiatkowski-Phillips-Schmidt-Shin (KPSS) tests. Additionally, ndiffs and nsdiffs functions are used to determine the number of non-seasonal and seasonal differences needed to make the time series stationary, respectively.

```r
# ARIMA identification and order determination
adf.test(training_set_4904_ts)
```

```
## Warning in adf.test(training_set_4904_ts): p-value smaller than printed p-value

##
##  Augmented Dickey-Fuller Test
##
## data:  training_set_4904_ts
## Dickey-Fuller = -5.1616, Lag order = 4, p-value = 0.01
## alternative hypothesis: stationary
```

```r
pp.test(training_set_4904_ts)
```

```
## Warning in pp.test(training_set_4904_ts): p-value smaller than printed p-value

##
##  Phillips-Perron Unit Root Test
##
## data:  training_set_4904_ts
## Dickey-Fuller Z(alpha) = -40.404, Truncation lag parameter = 3, p-value
## = 0.01
## alternative hypothesis: stationary
```

```r
kpss.test(training_set_4904_ts)
```

```
## Warning in kpss.test(training_set_4904_ts): p-value greater than printed
## p-value

##
##  KPSS Test for Level Stationarity
##
## data:  training_set_4904_ts
## KPSS Level = 0.23935, Truncation lag parameter = 3, p-value = 0.1
```

```r
# seasonal differencing
ndiffs(training_set_4904_ts)
```

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

```r
nsdiffs(training_set_4904_ts)
```

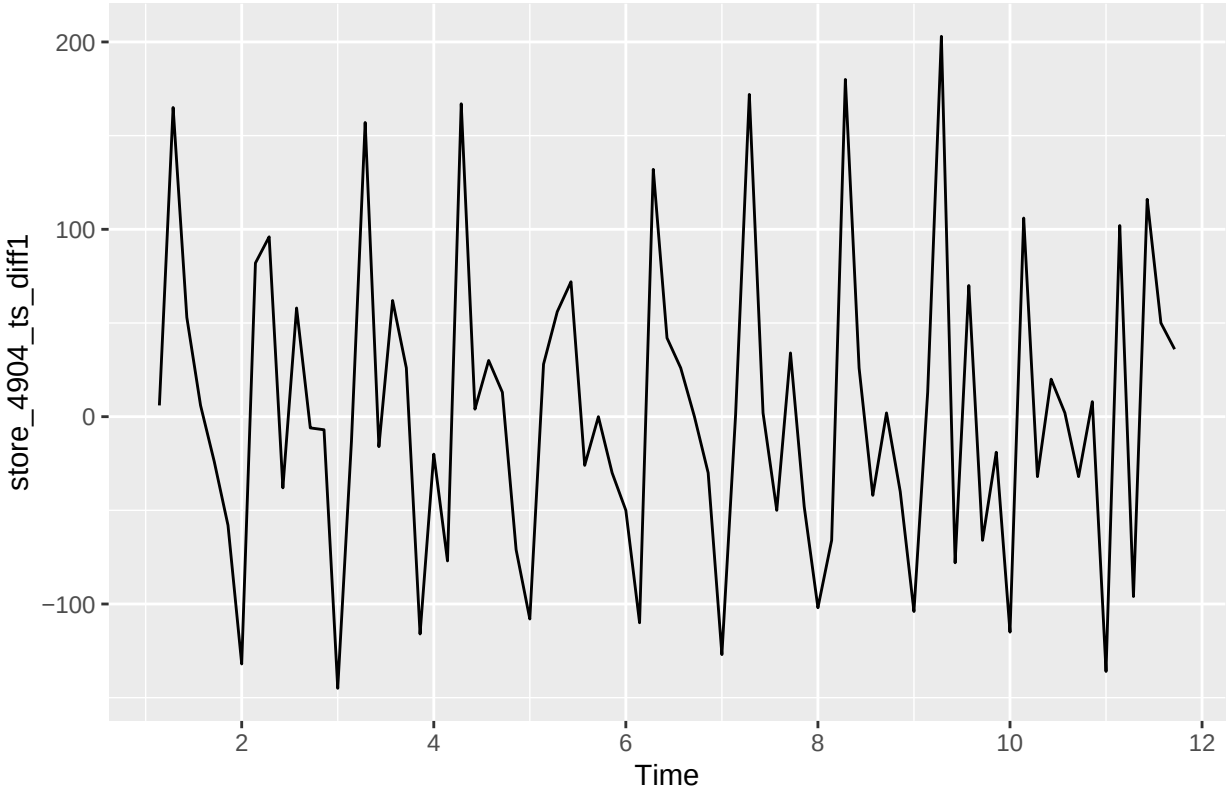
```
## [1] 1
```

Key insights:

1. Both the Augmented Dickey-Fuller Test and the Phillips-Perron Unit Root Test suggest stationarity with low p-values, indicating that the time series data for store 4904 is likely stationary.
2. The KPSS Test for Level Stationarity shows a p-value greater than 0.05, suggesting that there is evidence against the null hypothesis of non-stationarity.
3. ndiff is 0 and nsdiff is 1, this suggests that a first-order seasonal difference is needed to achieve stationarity in your time series data.

   In other words, differencing the data once with respect to the seasonal component should be sufficient to make the series stationary. Hence, we carry out differencing and plot our resulted data:

```r
# ARIMA identification and order determination (continued)
store_4904_ts_diff1 <- diff(training_set_4904_ts, differences = 1)
autoplot(store_4904_ts_diff1)
```



Key insight: After differencing, data appears to be stationary now.

Now we recheck the required results through the same tests again:

```r
# ARIMA identification and order determination
adf.test(store_4904_ts_diff1)
```

```
## Warning in adf.test(store_4904_ts_diff1): p-value smaller than printed p-value

##
##  Augmented Dickey-Fuller Test
##
## data:  store_4904_ts_diff1
## Dickey-Fuller = -10.876, Lag order = 4, p-value = 0.01
## alternative hypothesis: stationary
```

```r
pp.test(store_4904_ts_diff1)
```

```
## Warning in pp.test(store_4904_ts_diff1): p-value smaller than printed p-value

##
##  Phillips-Perron Unit Root Test
##
## data:  store_4904_ts_diff1
## Dickey-Fuller Z(alpha) = -71.063, Truncation lag parameter = 3, p-value
## = 0.01
## alternative hypothesis: stationary
```

```r
kpss.test(store_4904_ts_diff1)
```

```
## Warning in kpss.test(store_4904_ts_diff1): p-value greater than printed p-value

##
##  KPSS Test for Level Stationarity
##
## data:  store_4904_ts_diff1
## KPSS Level = 0.046695, Truncation lag parameter = 3, p-value = 0.1
```

```r
# seasonal differencing
ndiffs(store_4904_ts_diff1)
```

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

```r
nsdiffs(store_4904_ts_diff1)
```

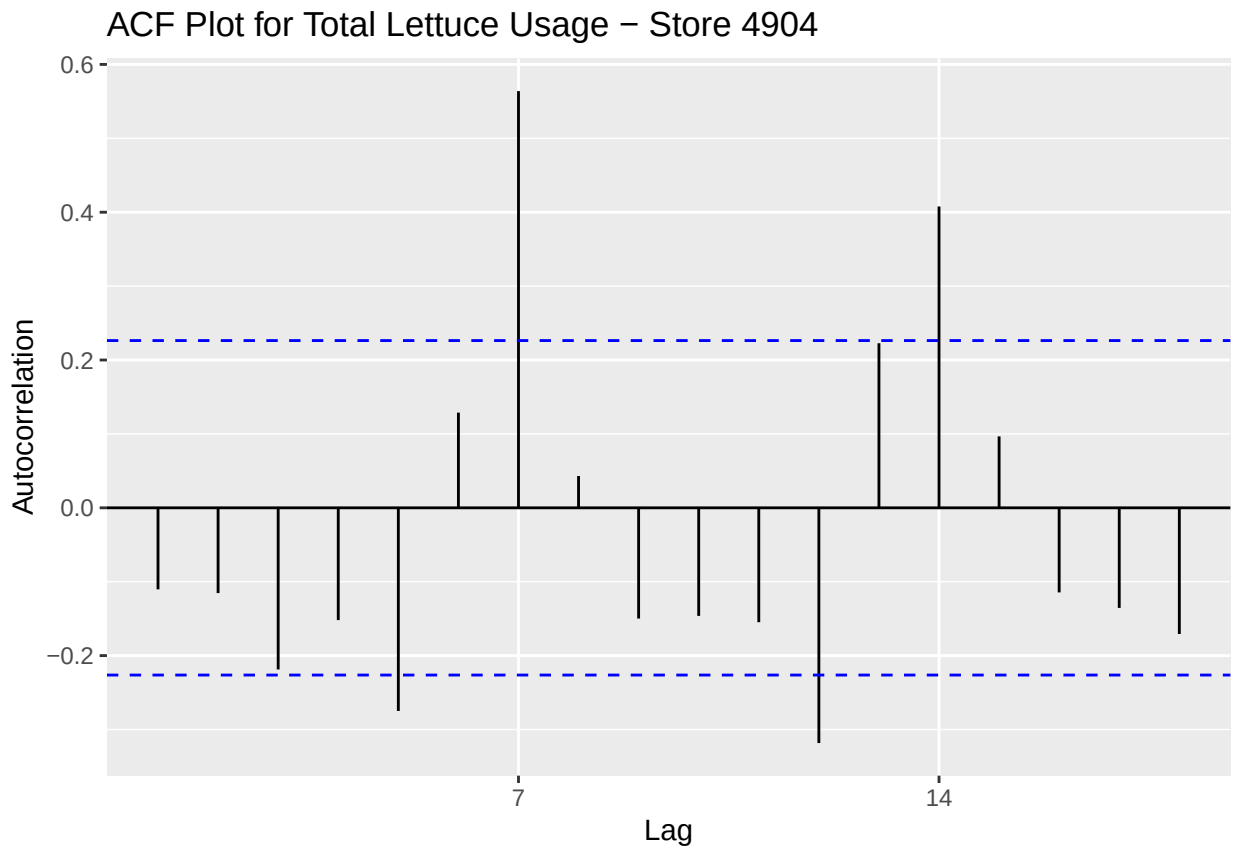
```
## [1] 0
```

The differenced time series, obtained through seasonal differencing, exhibits strong evidence of stationarity based on Augmented Dickey-Fuller and Phillips-Perron tests, supporting its suitability for further modeling and forecasting.

To further solidfy the selection of appropriate ARIMA parameters for our time series, we turn to the analysis of the autocorrelation function (ACF) and partial autocorrelation function (PACF). The visual examination of ACF and PACF plots aids in determining the optimal order for the ARIMA model, facilitating a more accurate and effective forecasting approach.

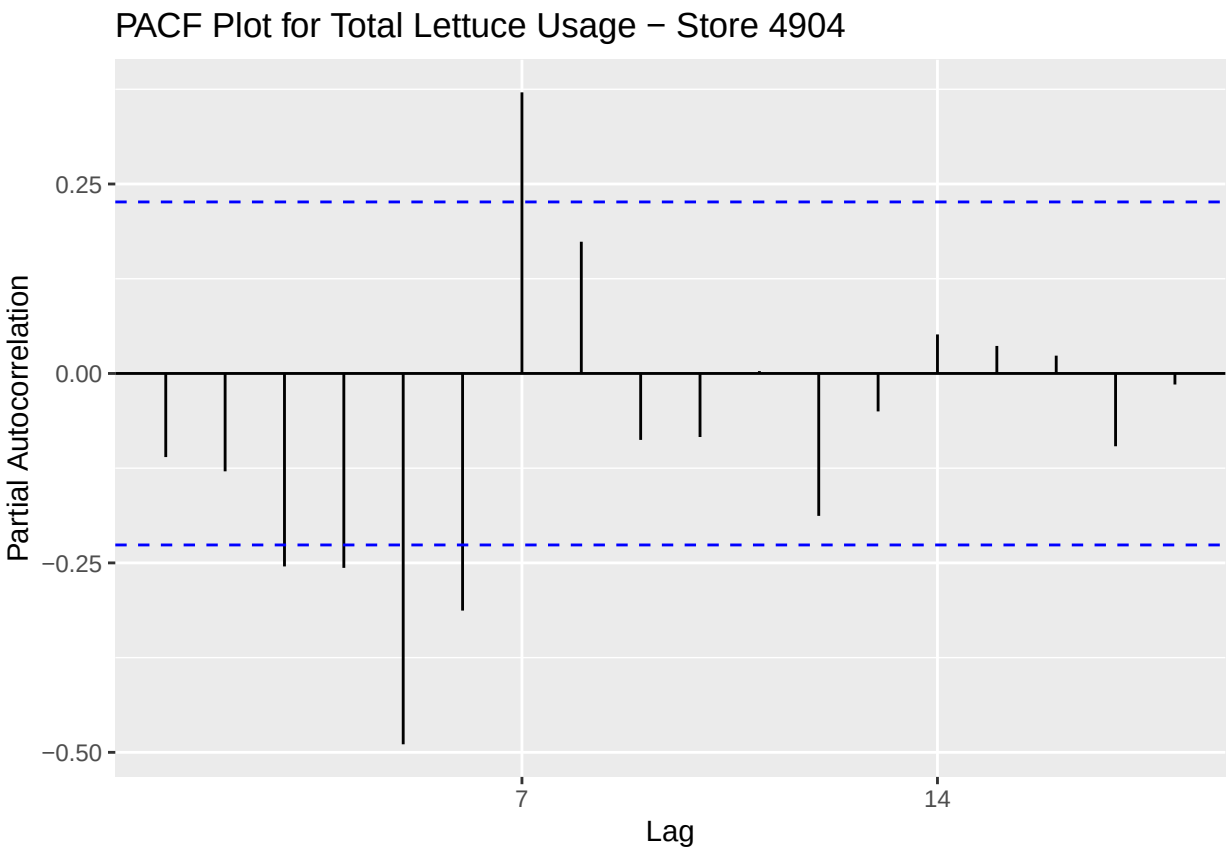
ACF Plot:

```r
# Determine p and q
ggAcf(store_4904_ts_diff1) +
  labs(title = "ACF Plot for Total Lettuce Usage - Store 4904",
       x = "Lag",
       y = "Autocorrelation")
```



PACF plot:

```r
ggPacf(store_4904_ts_diff1) +
  labs(title = "PACF Plot for Total Lettuce Usage - Store 4904",
       x = "Lag",
       y = "Partial Autocorrelation")
```



Model Selection:

The auto.arima function was used to explore various combinations of differencing and seasonal parameters, considering the Akaike Information Criterion (AIC) as the selection criterion. Upon analysing the AIC values, three models with the lowest AICs were chosen. Subsequently, we proceeded to fit these selected models on our training data to further evaluate their forecasting capabilities.

(The code line for generating the auto arima models 'auto_arima_result_4904' is commented because of the long series of output. However, it stays uncommented in the RMD file)

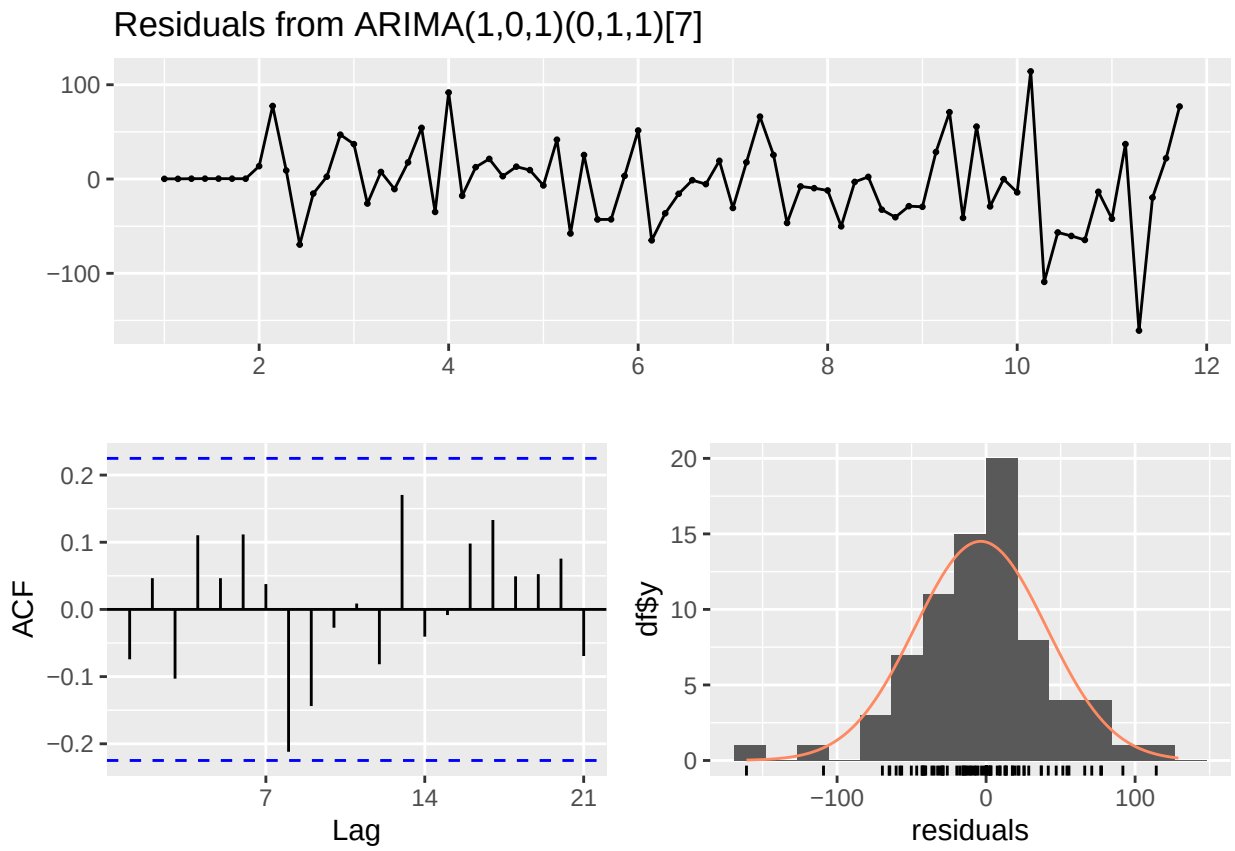
```
# Automatic ARIMA model selection
#auto_arima_result_4904 <- auto.arima(training_set_4904_ts, ic = 'aicc',
#stepwise = FALSE, approximation = FALSE, d=0, D = 1, trace = TRUE)

#auto_arima_result_4904

#ARIMA(1,0,1)(0,1,1)[7]: 736.5686
#ARIMA(1,0,3)(0,1,1)[7]: 737.3794
#ARIMA(2,0,1)(0,1,1)[7]: 738.3387


# Fit the ARIMA model
arima_model_4904.m1 <- Arima(training_set_4904_ts, order = c(1, 0, 1),
seasonal = list(order = c(0, 1, 1), period = 7), include.drift = FALSE)
arima_model_4904.m2 <- Arima(training_set_4904_ts, order = c(1, 0, 3),
seasonal = list(order = c(0, 1, 1), period = 7), include.drift = FALSE)
arima_model_4904m3 <- Arima(training_set_4904_ts, order = c(2, 0, 1),
```

```
seasonal = list(order = c(0, 1, 1), period = 7), include.drift = FALSE)
```

Residuals analysis:

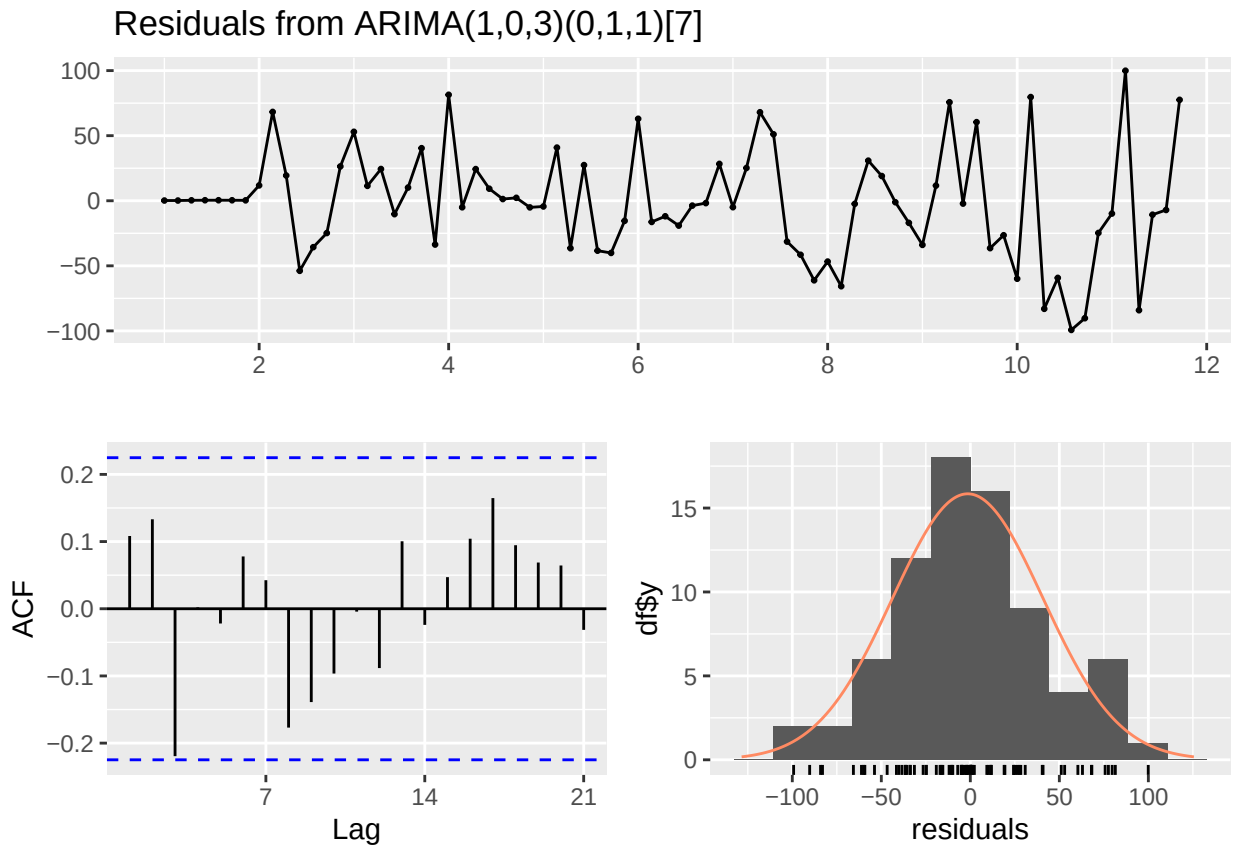
To make sure that the models we selected effectively represent the temporal trends, we used residual analysis. For residuals, R's checkresiduals() function offers an extensive collection of statistical tests and diagnostic charts. In order to confirm the accuracy of our projections and to make appropriate judgments, this phase is essential.

```
# Residual analysis
checkresiduals(arima_model_4904.m1)
```



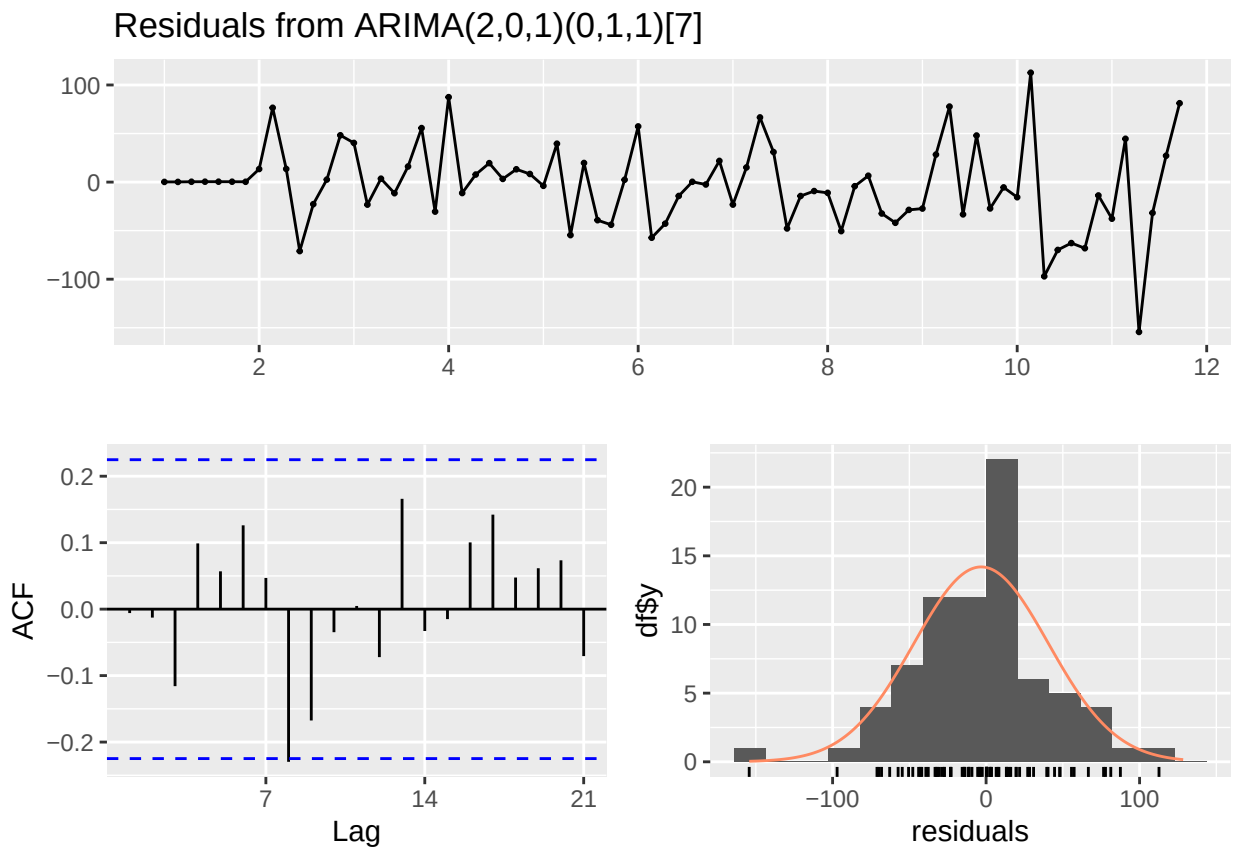

```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(1,0,1)(0,1,1)[7]
## Q* = 13.154, df = 11, p-value = 0.2834
##
## Model df: 3.   Total lags used: 14
```

```
checkresiduals(arima_model_4904.m2)
```




```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(1,0,3)(0,1,1)[7]
## Q* = 13.965, df = 9, p-value = 0.1236
##
## Model df: 5.   Total lags used: 14
```

```
checkresiduals(arima_model_4904m3)
```




```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(2,0,1)(0,1,1)[7]
## Q* = 14.086, df = 10, p-value = 0.1691
##
## Model df: 4.   Total lags used: 14
```

Forecasting and Accuracy:

Considering the fitted ARIMA models, we produce forecasts for the validation set in this stage. The validation set is a certain fraction of our time series data that was not used for model training. The 'accuracy' function, which offers a comprehensive set of parameters to evaluate the model's performance against the real data, is then used to determine how accurate these forecasts are. This analysis helps determine which ARIMA model is most likely to accurately estimate lettuce demand at Store 4904.

```
# Forecast using the fitted model
arima_forecast_4904.m1 <- forecast(arima_model_4904.m1, h = length(validation_set_4904))
arima_forecast_4904.m2 <- forecast(arima_model_4904.m2, h = length(validation_set_4904))
arima_forecast_4904.m3 <- forecast(arima_model_4904m3, h = length(validation_set_4904))


# Model evaluation
accuracy(arima_forecast_4904.m1, validation_set_4904)
```

```
##                     ME      RMSE       MAE       MPE      MAPE      MASE
```

```
## Training set -3.593391 44.06127 31.96752 -2.41889 10.80270 0.7459449
## Test set     47.100844 75.64057 56.36357 13.58546 17.26304 1.3152135
##                     ACF1 Theil's U
## Training set -0.07424073        NA
## Test set     -0.30952322 0.8686175
```

```
accuracy(arima_forecast_4904.m2, validation_set_4904)
```

```
##                     ME     RMSE      MAE       MPE     MAPE      MASE
## Training set -1.452362 42.08627 31.57496 -1.218082 10.23758 0.7367847
## Test set     18.601654 58.81662 45.24012  4.275120 14.75903 1.0556537
##                    ACF1 Theil's U
## Training set  0.1082813        NA
## Test set     -0.2831305 0.6727696
```

```
accuracy(arima_forecast_4904.m3, validation_set_4904)
```

```
##                     ME     RMSE      MAE       MPE     MAPE      MASE
## Training set -3.216025 43.69884 31.94161 -2.212049 10.71236 0.7453402
## Test set     44.766491 71.40238 52.77185 13.003111 16.11484 1.2314027
##                     ACF1 Theil's U
## Training set -0.00589713        NA
## Test set     -0.28482908 0.8206349
```

```
#best rmse value is for arima_forecast_4904.m2 model
```

Key Insights:

1. With the lowest RMSE on the test sets, Model 2 performs better than the others according to the evaluation metrics, indicating that it is capable of forecasting lettuce demand at Store 4904 with greater accuracy.
2. Comparing this to models m1 and m3, a smaller RMSE demonstrates better performance in minimizing the prediction error.
3. The smaller RMSE values indicate that Model 2's forecasts align more closely with the actual values, making it the preferred choice for lettuce demand prediction in this context.

Fitting the best ARIMA Model on the entire data set:

The selected ARIMA(1,0,3)(0,1,1)[7] model, which has been determined to be the best-performing model on the training and validation sets, is being fitted to the whole dataset for Store 4904 in this stage of development. Establishing a definitive forecast for the subsequent 14 days is the ultimate objective.

```
#now fit the first best model on our entire dataset
arima_model_4904_final <- Arima(store_series_ts, order = c(1, 0, 3), seasonal = list(order = c(0, 1, 1)

# Final forecast
final_forecast_4904 <- forecast(arima_model_4904_final, h = 14)

# Extract point forecasts
forecast_values_4904 <- final_forecast_4904$mean

# Create forecast dates from 16/06/2015 to 29/06/2015
```

```r
forecast_dates_4904 <- seq(as.Date("2015-06-16"), by = "days", length.out = 14)

# Create a data frame with dates and point forecasts
final_forecast_data_4904 <- data.frame(date = forecast_dates_4904, forecast = forecast_values_4904)

# Print the final forecast data
print(final_forecast_data_4904)
```

```
##          date forecast
## 1  2015-06-16 359.0667
## 2  2015-06-17 343.3956
## 3  2015-06-18 305.7828
## 4  2015-06-19 220.9437
## 5  2015-06-20 222.4549
## 6  2015-06-21 347.0936
## 7  2015-06-22 341.8644
## 8  2015-06-23 361.4128
## 9  2015-06-24 346.4233
## 10 2015-06-25 308.5037
## 11 2015-06-26 223.5323
## 12 2015-06-27 224.9176
## 13 2015-06-28 349.4366
## 14 2015-06-29 344.0934
```

Part 1 Conclusion:

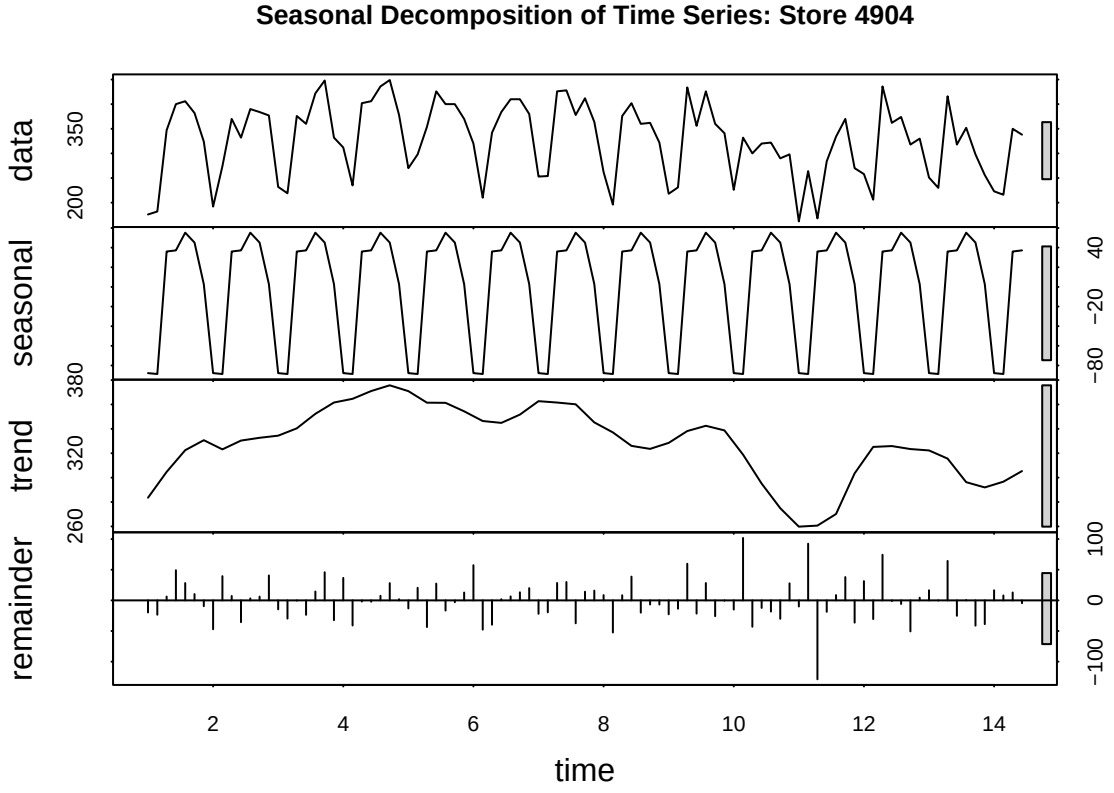
In summary, the model has been validated on historical data, and its accuracy assessed through metrics such as RMSE. We built a forecast for the next 14 days using this ARIMA model, which offers insightful information for inventory management.

Part 2) Holt Winters Model: Moving forward, we will extend our analysis by investigating the Holt-Winters model in order to determine the best method of forecasting the demand for lettuce in our dataset.

Seasonal Decomposition:

Decomposing a time series facilitates the process of recognizing and illustrating the underlying patterns in the data, including the general trend and recurring seasonal patterns. Determining the total variability in the time series is made easier by having a clear understanding of each component's contribution

```r
plot(stl(store_series_ts, s.window = "period"), main="Seasonal Decomposition of Time Series: Store 4904
```



Key Insights:

1. The trend is not linear here. It exhibits wavy patterns with fluctuations, rather than following a strict linear trajectory.
2. An additive seasonal component and an additive error seems to take place in the time-series and will be needed to the upcoming forecast model.

Model selection:

I performed ETS (Exponential Smoothing State Space) forecasting using the training set for Store 4904. First, I used the ets() function without specifying a model (store_4904_ets_training2) to identify the best ETS model for the given data. I got (A, A, A) model as the best model in this case. According to my analysis in the time series decomposition, the trend of the data seems to be non-linear. This indicates that the linearity is absent. Hence, I will choose to go ahead with the ANA Model for this case. and I used this model (store_4904_ets_training) on the training dataset for further analysis.

```
# ETS forecast based on the training set
store_4904_ets_training2 <- ets(training_set_4904_ts, model='ZZZ')

# ETS MNM forecast based on the training set
store_4904_ets_training <- ets(training_set_4904_ts, model='ANA', ic = 'aicc')
store_4904_ets_training
```

```
## ETS(A,N,A)
##
## Call:
```

```
##  ets(y = training_set_4904_ts, model = "ANA", ic = "aicc")
##
##   Smoothing parameters:
##     alpha = 0.1935
##     gamma = 1e-04
##
##   Initial states:
##     l = 335.4537
##     s = 10.9948 54.9311 56.7306 38.9975 20.9684 -86.7199
##           -95.9024
##
##   sigma:  43.9461
##
##      AIC     AICc      BIC
## 914.5671 917.9517 937.8744
```

In-sample estimation error:

Here we are evaluating the in-sample estimation error of the ETS (Error, Trend, Seasonal) model using the accuracy() function, which provides metrics like Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and others.

```
# In-sample estimation error for ETS
in_sample_errors_ets <- accuracy(store_4904_ets_training)
in_sample_errors_ets
```

```
##                      ME     RMSE      MAE       MPE     MAPE      MASE
## Training set -3.960539 41.26204 31.63586 -3.438702 10.98139 0.7382057
##                      ACF1
## Training set -0.08936168
```

In conclusion, these metrics provide insights into the performance of the ETS model on the training data.

Out-of-sample evaluation:

I am using the trained ETS model (store_4904_ets_training) to generate forecasts for the length of the validation set. Furthermore, I evaluate the accuracy of the ETS model's forecasts against the actual values in the validation set. This helps in understanding how well the ETS model performs on unseen data by comparing its forecasts with the actual values in the validation set, using a range of accuracy metrics.

```
#Out-of-sample evaluation for ETS
store_4904_ets_forecast <- forecast.ets(store_4904_ets_training, h = length(validation_set_4904))

# Forecasting erros - ETS MODEL
out_of_sample_errors_ets <- accuracy(store_4904_ets_forecast, validation_set_4904)
out_of_sample_errors_ets
```

```
##                      ME     RMSE      MAE       MPE     MAPE      MASE
## Training set -3.960539 41.26204 31.63586 -3.438702 10.98139 0.7382057
## Test set     34.049448 55.95950 44.43286 10.551570 14.25775 1.0368169
##                      ACF1 Theil's U
## Training set -0.08936168        NA
## Test set     -0.02417914 0.6328567
```

Building the ANA model on the entire data set:

The trained model is now employed to forecast the future values for the entire time series (store_series_ts). The model is trained on the entire dataset, and then we generate a forecast for the next 14 time points (h = 14).

```
# ETS MNM forecast on the entire dataset
final_ets_model <- ets(store_series_ts, model ='ANA', ic = 'aicc')
final_ets_forecast <- forecast.ets(final_ets_model, h = 14)

# Extract point forecasts
forecast_values_ets <- final_ets_forecast$mean

# Create forecast dates from 16/06/2015 to 29/06/2015
forecast_dates_ets <- seq(as.Date("2015-06-16"), by = "days", length.out = 14)

# Create a data frame with dates and point forecasts
final_forecast_data_ets <- data.frame(date = forecast_dates_ets, forecast = forecast_values_ets)

# Print the final forecast data
print(final_forecast_data_ets)
```

```
##          date forecast
## 1  2015-06-16 359.2945
## 2  2015-06-17 347.8354
## 3  2015-06-18 308.8660
## 4  2015-06-19 212.2648
## 5  2015-06-20 212.9511
## 6  2015-06-21 336.5967
## 7  2015-06-22 336.8785
## 8  2015-06-23 359.2945
## 9  2015-06-24 347.8354
## 10 2015-06-25 308.8660
## 11 2015-06-26 212.2648
## 12 2015-06-27 212.9511
## 13 2015-06-28 336.5967
## 14 2015-06-29 336.8785
```

Part 2 Conclusion:

The resulting 'final_ets_forecast' contains the forecasted values and allows me to project future trends and seasonality based on the characteristics learned from the historical data.

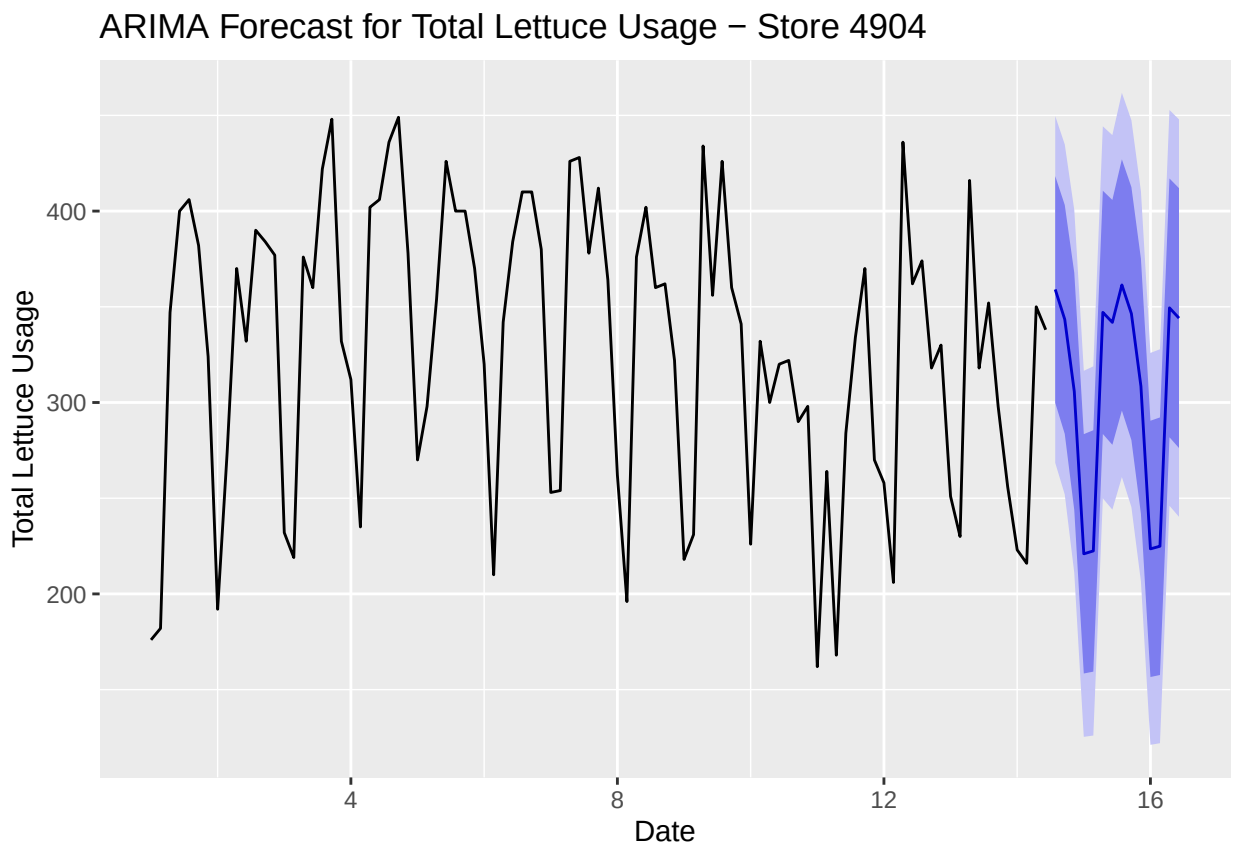
Part 3 Comparison: Now, that we have both the Arima and the Holt winters model setup, we need to assess the forecasting performance of both the ARIMA and Holt-Winters models for a two-week timeframe.

To gain insights into the accuracy of each model's predictions, we initiated the examination by visualizing the forecasted values individually for ARIMA and Holt-Winters. This step allows us to observe the pattern and behavior of the forecasted series generated by each model.

A) Plot the forecasts of the ARIMA Model:

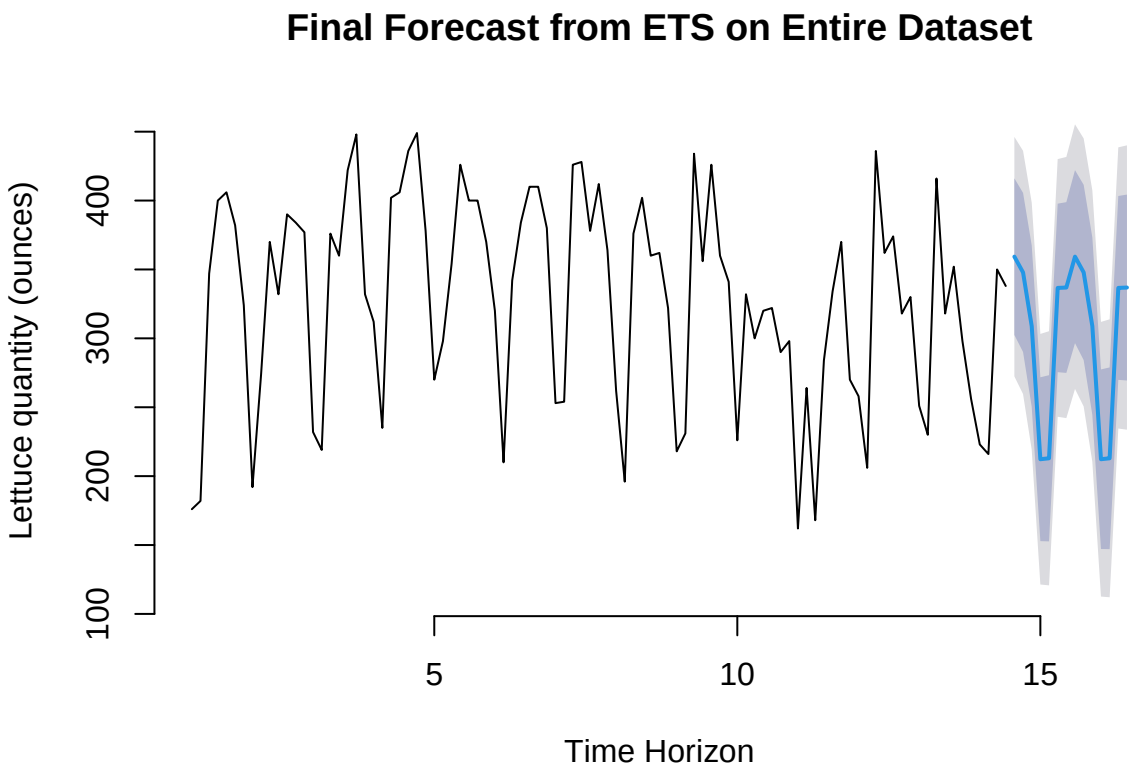
```
# Plot the forecast
autoplot(final_forecast_4904) +
```

```
  labs(title = "ARIMA Forecast for Total Lettuce Usage - Store 4904",
       x = "Date",
       y = "Total Lettuce Usage")
```



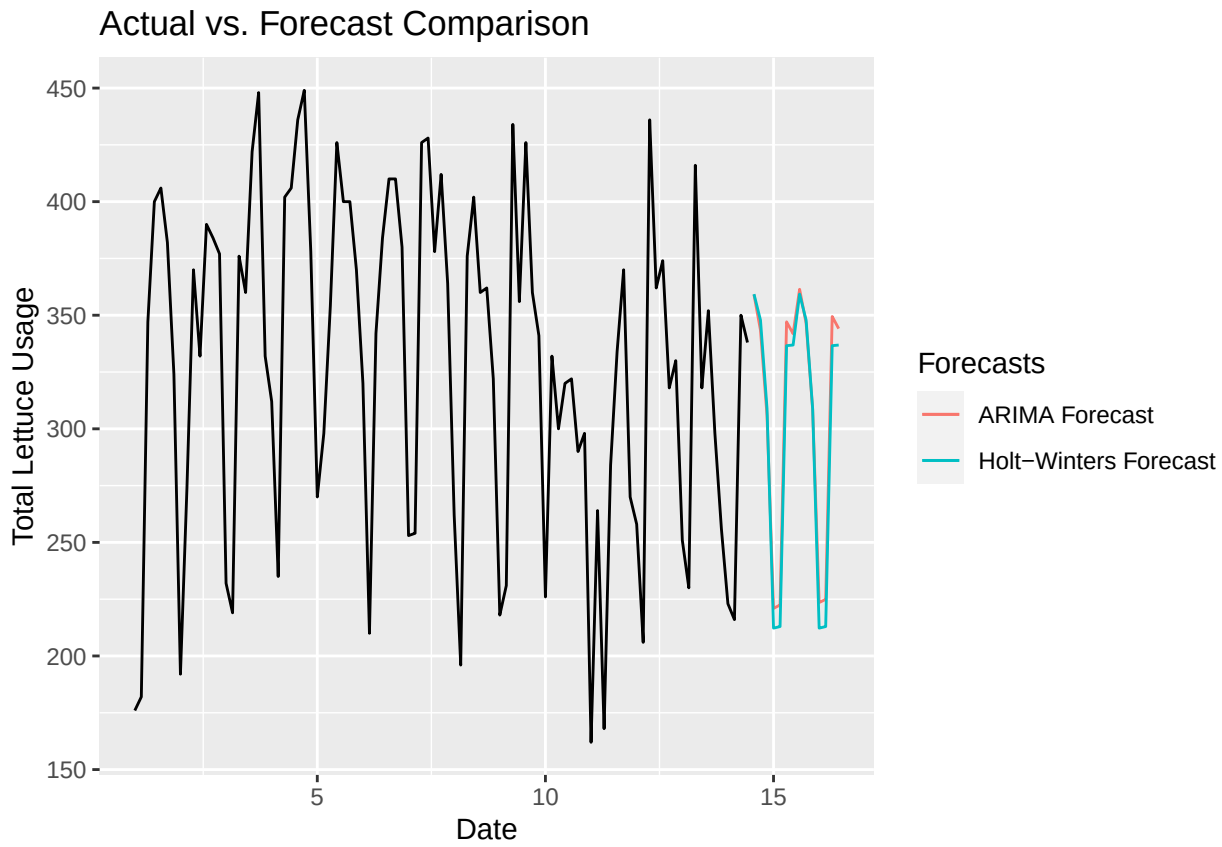
B) Plot the forecasts of the Holt Winter Model:

```
# Plot the final forecast
plot(final_ets_forecast, main = "Final Forecast from ETS on Entire Dataset", xlab="Time Horizon", ylab=
```



C) ARIMA vs HoltWinters Model Forecasts:

```
# Plot actual data
autoplot(store_series_ts) +
  labs(title = "Actual vs. Forecast Comparison",
       x = "Date",
       y = "Total Lettuce Usage") +
  autolayer(final_forecast_4904$mean, series = "ARIMA Forecast") +
  autolayer(final_ets_forecast$mean, series = "Holt-Winters Forecast") +
  guides(color = guide_legend(title = "Forecasts"))
```



Analysis of the Accuracies of both the Models:

Following the visual analysis of the ARIMA and Holt-Winters forecasts, we proceeded to a quantitative assessment using the Root Mean Squared Error (RMSE). RMSE is a widely utilized metric for evaluating the accuracy of forecasting models, providing a comprehensive measure of the differences between predicted and observed values. By calculating the RMSE for both the ARIMA and Holt-Winters models, we aim to quantify the level of accuracy each model achieves in capturing the actual values. A lower RMSE signifies better predictive performance, indicating that the model's forecasts closely align with the observed data. This numerical comparison aids in selecting the model that exhibits superior accuracy and reliability in forecasting the daily demand for lettuce in our dataset over the specified two-week period.

```
#ARIMA model
accuracy(arima_forecast_4904.m2, validation_set_4904)
```

```
##                      ME     RMSE      MAE       MPE     MAPE      MASE
## Training set -1.452362 42.08627 31.57496 -1.218082 10.23758 0.7367847
## Test set     18.601654 58.81662 45.24012  4.275120 14.75903 1.0556537
##                    ACF1 Theil's U
## Training set  0.1082813        NA
## Test set     -0.2831305 0.6727696
```

```
#ets ana model
out_of_sample_errors_ets
```

```
##                      ME     RMSE      MAE       MPE     MAPE      MASE
## Training set -3.960539 41.26204 31.63586 -3.438702 10.98139 0.7382057
```

```
## Test set     34.049448 55.95950 44.43286 10.551570 14.25775 1.0368169
##                     ACF1 Theil's U
## Training set -0.08936168        NA
## Test set     -0.02417914 0.6328567
```

RMSE Values	ARIMA	Holt winters
Train data	42.08627	41.26204
test data	58.81662	55.95950

Key Insights:

ETS model, the ANA Model, has lower RMSE values for both the training and test sets compared to the ARIMA model. This suggests that the ETS model provides a better fit to the data in terms of forecasting accuracy. The lower RMSE values indicate that the ETS model's predictions are closer to the actual values, both in the training and test phases.

In summary, based on the RMSE values, the ETS model outperforms the ARIMA model in this specific analysis.

Final Forecasted Values:

Hence, the final predictions for the Store 4904 using the ETS - ANA model are:

```r
# Print the final forecast data
print(final_forecast_data_ets)
```

```
##          date forecast
## 1  2015-06-16 359.2945
## 2  2015-06-17 347.8354
## 3  2015-06-18 308.8660
## 4  2015-06-19 212.2648
## 5  2015-06-20 212.9511
## 6  2015-06-21 336.5967
## 7  2015-06-22 336.8785
## 8  2015-06-23 359.2945
## 9  2015-06-24 347.8354
## 10 2015-06-25 308.8660
## 11 2015-06-26 212.2648
## 12 2015-06-27 212.9511
## 13 2015-06-28 336.5967
## 14 2015-06-29 336.8785
```