Enhancing Traditional Time Series Forecasts with Machine Learning and Stacked Ensemble Methods

Todd Blackwell

2024-05-17

Table of Contents

Introduction

In this real-life example, a company that owns multiple restaurants seeks to understand if Machine Learning (ML) can improve upon their traditional time series sales forecasting models.

Overview

The primary objective is to determine whether ML algorithms can improve the accuracy and effectiveness of existing forecasting methodologies. If we achieve this objective we can empower company leaders to make better forecasts and better business decisions.

We will use the concepts and skills learned throughout the Harvard Data Science courses to tackle this real-world data challenge. The concepts and skills we will use include R, data wrangling, data visualization, algorithm building, productivity, statistics, and machine learning.

Executive Summary

The company is currently using a linear regression (LR) method for sales forecasting. This LR model use daily historical sales at the store level and other relevant predictive variables such as day of week, seasonality, holidays and paydays to predict future sales trends. The LR model assumes a linear relationship between the predictor variables and the target variable (sales). The LR model estimates the coefficients (influence) that each predictor has on sales. The LR model offers simplicity and is easy to interpret, but struggles to capture complex nonlinear relationships, limiting the predictive accuracy.

We attempt to improve upon the LR model by adding other relevant third-party predictor variables such as temperature, gas prices, consumer price index (CPI), and unemployment.

We also create other traditional time series models such as Autoregressive Integrated Moving Average (ARIMA), Seasonal-Trend Decomposition with Loess Forecasting (STLF), and Prophet, a newer time-series forecaster from Facebook. The evaluation metrics from these traditional models are found below:

Summary of Models

Evaluation_Metric	Naive	Linear	ARIMA	STLF	PROPHET
RMSE	3217.6101	1623.1146	1862.3678	2906.0822	1885.4627
MAPE	44.98366	31.32334	33.25723	39.69940	46.66521
Forecast Bias	2650.0092	0.00000	212.04841	2239.8491	-885.17100
Forecast Accuracy	55.91194	78.32954	76.23366	61.87877	71.81258

We find that the Linear, ARIMA and Prophet models are a significant improvement relative to the naive baseline model. However, the STLF model is not significantly better than the naive forecast, and we do not recommend using the STLF model going forward.

Next, we implement two ML models, Random Forest (RF) and Gradient Boosting Machines (GBM).

Summary of Models

Evaluation_Metric	Naive	RANDOM_FOREST	GBM
RMSE	3217.61	1107.7476	1336.88
MAPE	44.9836	19.75736	22.6558
Forecast Bias	2650.00	190.72133	0.11525
Forecast Accuracy	55.9119	21.86108	16.8109

We find that in comparison to the traditional models, both ML models narrow the deviations of the predictive values, and that the GBM model has minimal bias. However the forecast accuracy of the ML models are much lower than the traditional models.

Next we pick three models to ensemble stack. Ensemble stacking in sales forecasting is a technique that involves combining multiple forecasting models to improve the accuracy of sales predictions. We choose to stack are the linear regression model, the ARIMA model, and the GBM ML model, and train this meta model using the linear, GBM and RF methods.

Summary of Models

Evaluation Metric	Stacked Linear	Stacked GBM	Stacked RF
RMSE	1118.10873	1013.85987	553.253532
MAPE	18.72952	16.29512	7.356060
Forecast Bias	0.00000	-20.63712	3.119288
Forecast Accuracy	85.07572	86.75844	93.451912

All three of the stacked ensemble methods improve the evaluation metrics relative to using a standalone model, with the stacked Random Forest model showing the most promise. Because the results of the ensemble methods are favorable, we decide to evaluate all three ensemble models on the unseen test dataset to assess their performance.

Summary of Models

Evaluation Metric	Stacked Linear	Stacked GBM	Stacked RF
RMSE	1157.32197	809.563257	270.234238
MAPE	18.91861	13.027538	4.063737
Forecast Bias	0.00000	1.628157	2.822545
Forecast Accuracy	84.94792	89.111282	96.258253

All three stacked ensemble methods perform well with the test dataset. The stacked LR testing model performs very similarly to the stacked LR training model. The stacked GBM model slightly improves with the testing data compared to the training data. The stacked RF model also improves with the testing data versus the training data. The stacked RF model performs significantly better than the other two stacked models.

Going forward, to improve the accuracy and the effectiveness of the sales forecast, we recommend that the company use the Stacked Ensemble Random Forest model for sales forecasting.

Methods/Analysis

Data Preparation

This section explains the processes and techniques used for data preparation, including data cleansing, data exploration and visualization, insights gained, and our modeling approach.

To begin, we load the necessary R libraries and we prepare the data. We take the sales, the features, and the stores; merge them, and finalize our data frame.

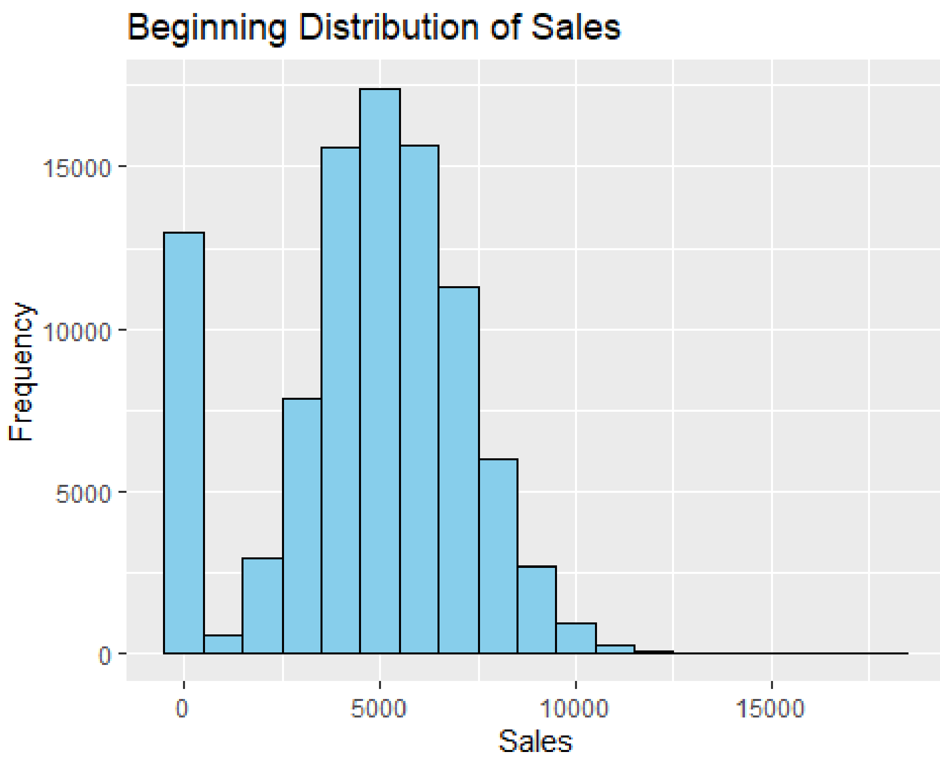
```r
# Examine structure of final_df
str(final_df)
```

```
## 'data.frame':    94181 obs. of  20 variables:
##  $ Store       : int  36530 36530 36530 36530 36530 36530 36530 36530 ...
##  $ Date        : Date, format: "2019-01-01" "2019-01-02" ...
##  $ Sales       : num  4978 6052 7130 7021 7479 ...
##  $ TAVG        : num  36 31.5 35 39.5 47.5 45 57 46.5 32.5 27 ...
##  $ GAS         : num  2.06 2.06 2.06 2.06 2.06 ...
##  $ CPI         : num  234 234 234 234 234 ...
##  $ UNEMPLOYMENT: num  3.5 3.5 3.5 3.5 3.5 3.5 3.5 3.5 3.5 3.5 ...
##  $ HOLIDAY     : int  1 0 0 0 0 0 0 0 0 0 ...
##  $ PAY         : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ MON         : int  0 0 0 0 0 0 1 0 0 0 ...
##  $ TUE         : int  1 0 0 0 0 0 0 1 0 0 ...
##  $ WED         : int  0 1 0 0 0 0 0 0 1 0 ...
##  $ THU         : int  0 0 1 0 0 0 0 0 0 1 ...
##  $ FRI         : int  0 0 0 1 0 0 0 0 0 0 ...
##  $ SAT         : int  0 0 0 0 1 0 0 0 0 0 ...
##  $ SUN         : int  0 0 0 0 0 1 0 0 0 0 ...
##  $ Region      : chr  "Midwest" "Midwest" "Midwest" "Midwest" ...
##  $ City        : chr  "Hannibal" "Hannibal" "Hannibal" "Hannibal" ...
##  $ State       : chr  "MO" "MO" "MO" "MO" ...
##  $ Zip.Code    : int  63401 63401 63401 63401 63401 63401 63401 63401 ...
```

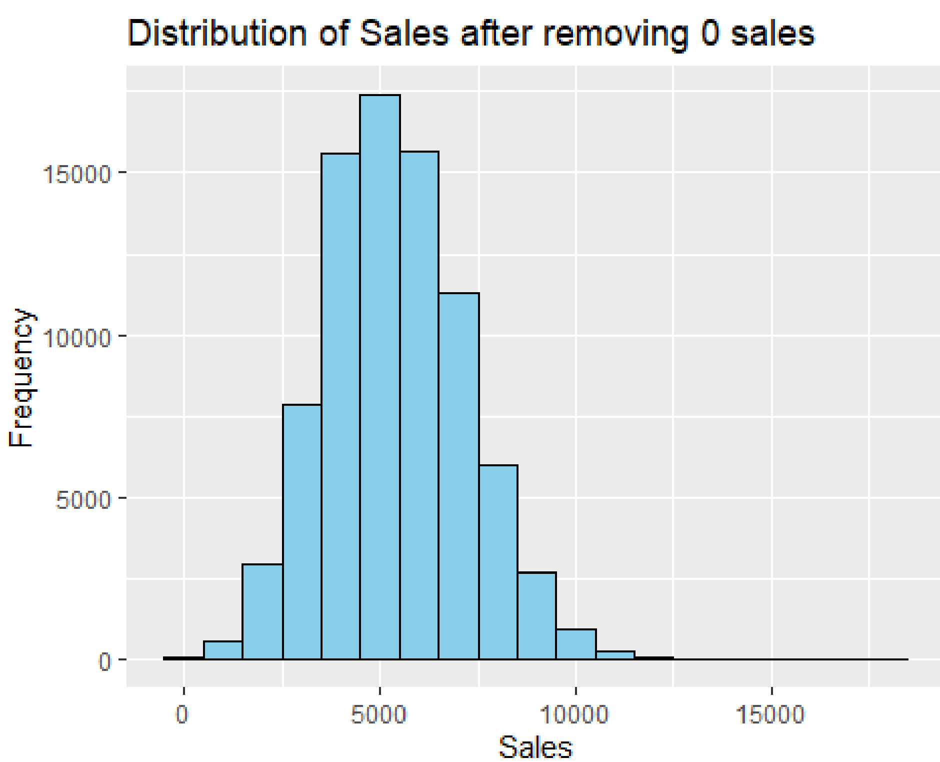
Data Exploration

In this section we are going to understand the data set, its structure and specificities.

First, we examine the distribution of sales.

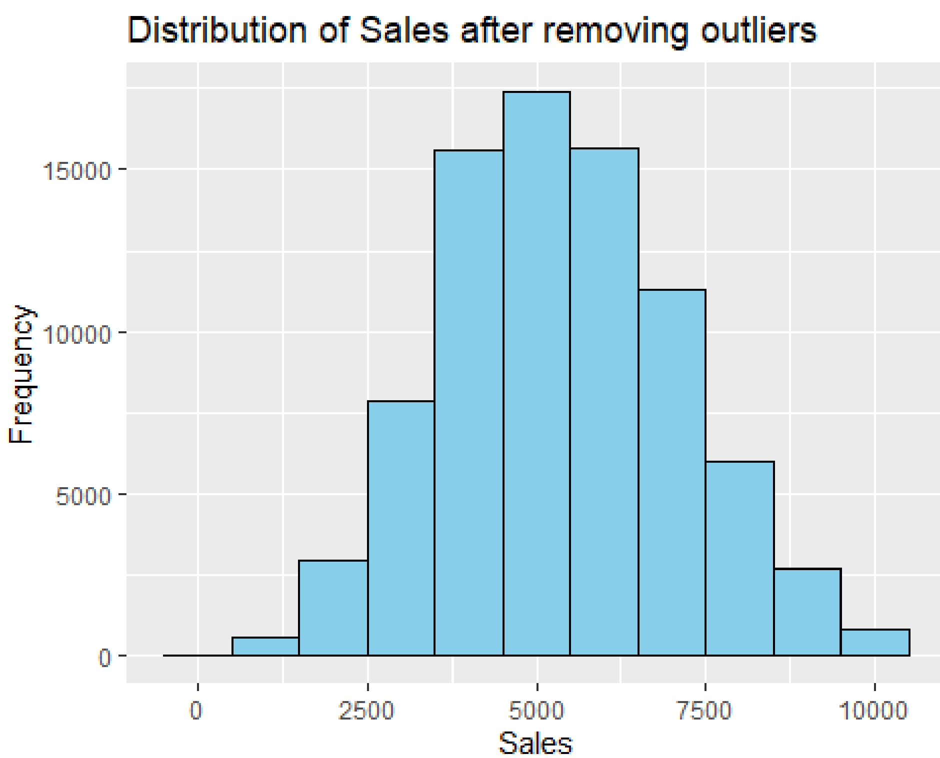


We see a large number of rows with zero values. Let's remove them.

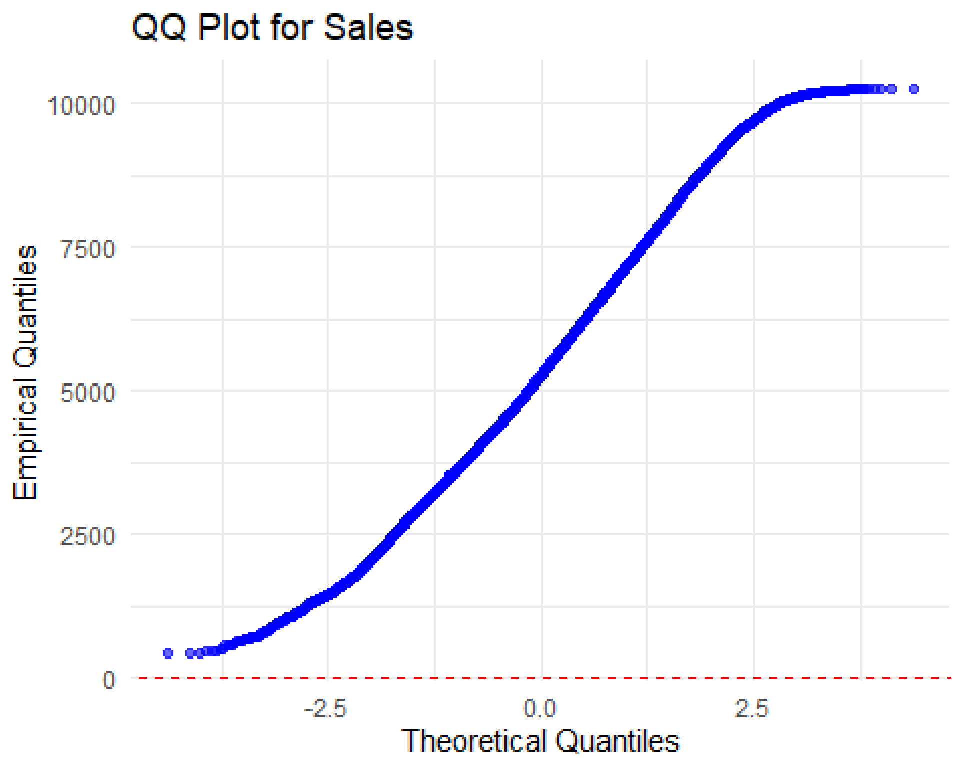


To further normalize the distribution, we remove outliers by determining the Interquartile Range (IQR).

```r
# Calculate the IQR (Interquartile Range) for Sales
Q1 <- quantile(final_df$Sales, 0.25)
Q3 <- quantile(final_df$Sales, 0.75)
IQR <- Q3 - Q1

# Define the upper and lower bounds for outliers
lower_bound <- Q1 - 1.5 * IQR
upper_bound <- Q3 + 1.5 * IQR

# Remove outliers
final_df <- final_df[final_df$Sales >= lower_bound & final_df$Sales <=
upper_bound, ]
```



Next, we generate theoretical quantiles, empirical quantiles, and a QQ plot to visually inspect whether the observed data matches the assumptions made about the distribution.



The points on the QQ plot closely follow a straight line, which suggests the assumed distribution is a good fit for the data.

Next, we calculate skewness and kurtosis to provide deeper insight into the shape and behavior of the distribution.

```
## Skewness: 0.18

## Kurtosis: 2.7
```

The interpretation of the Skewness value of 0.18 indicates a slight positive skew, meaning the data is slightly skewed to the right.

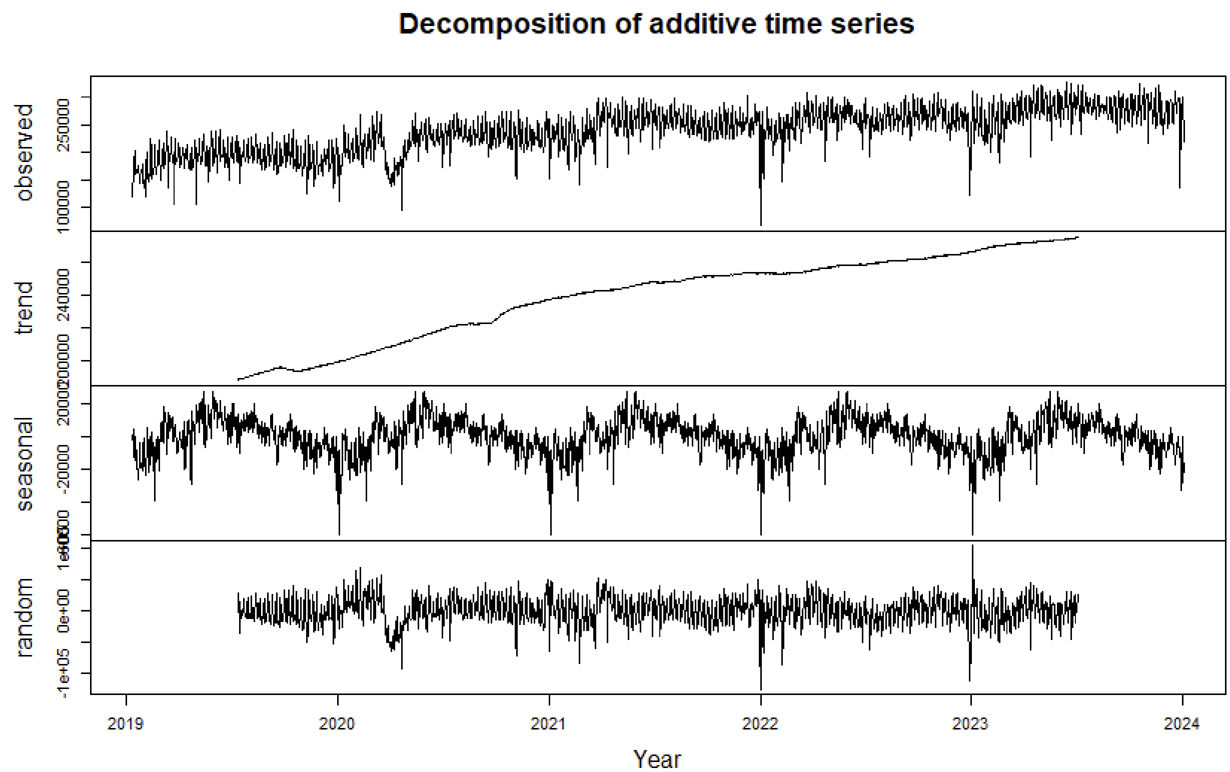
A Kurtosis value of 2.7 suggests that the distribution has fewer outliers and is slightly less peaked than a perfectly normal distribution, which would have a Kurtosis of 3.

Feature Engineering

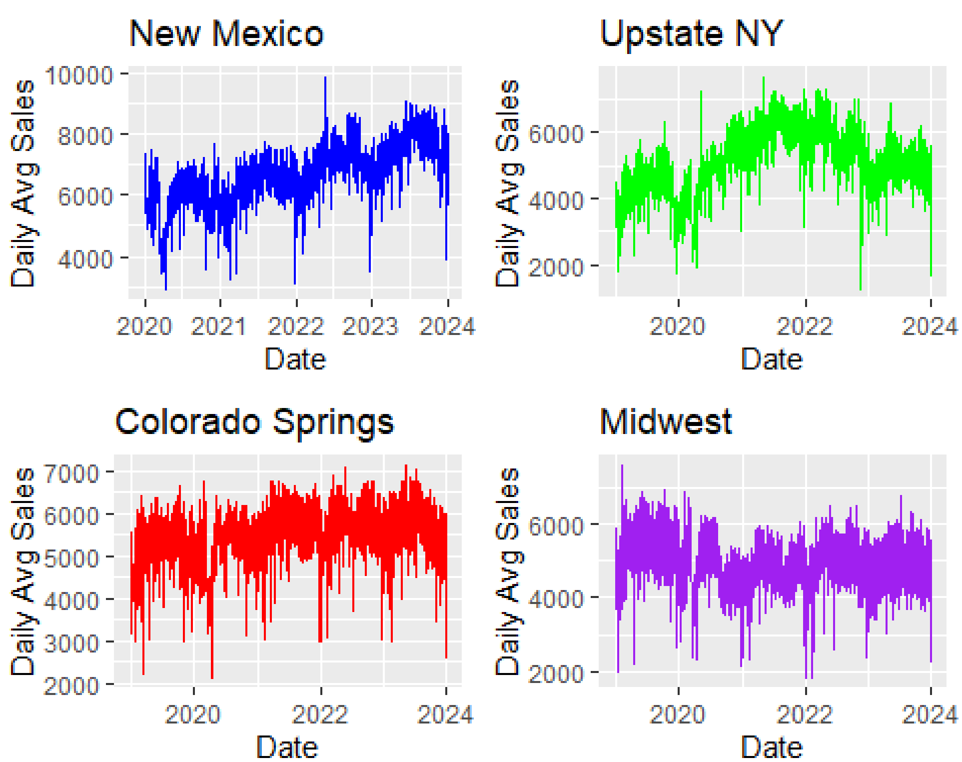
In this section, we are going to create informative features to capture the effects of all phenomenon that could impact sales.

Seasonality

We believe that there is seasonality in our sales data. To validate this, we aggregate the sales across all stores, create a time series object for total sales, perform an additive decomposition, then extract and plot the seasonal components. Our plot validates there is sales seasonality, which we will factor into our forecast models.



We believe that seasonality and volume may differ by geographical region. To validate this, we plot the daily aggregated sales for each region. Indeed, seasonality and volume vary by region. Therefore regions will be one of our predictors of sales.



Date

We engineer the date to create temporal features that capture information related to time.

```r
############################
# Feature Engineering - Date
############################

#Create Temporal Features from 'Date'
final_df$Month <- month(final_df$Date)
final_df$Day <- day(final_df$Date)
final_df$Quarter <- quarter(final_df$Date)
final_df$DayOfWeek <- wday(final_df$Date)
```

Temperature

We engineer the temperature by determining the daily temperature difference from the mean temperature. This will prove helpful in determining if temperature plays a role in sales volume.

```r
#############################
# Feature Engineering - Temps
#############################

# Calculate the mean of TAVG
mean_TAVG <- mean(final_df$TAVG, na.rm = TRUE)

# Calculate the difference from the mean for TAVG
final_df$TAVG_diff_from_mean <- final_df$TAVG - mean_TAVG
```

Consumer Price Index (CPI)

We engineer the CPI in a similar way, the difference from the mean, in an effort to determine if change in consumer prices play a role in sales volume.

```r
############################
# Feature Engineering - CPI
############################

# Calculate the mean of CPI
mean_CPI <- mean(final_df$CPI, na.rm = TRUE)

# Calculate the difference from the mean for CPI
final_df$CPI_diff_from_mean <- final_df$CPI - mean_CPI
```

Unemployment

Similarly, we engineer the difference in the unemployment rate versus the mean to determine if unemployment plays a role in sales.

```r
#####################################
# Feature Engineering - Unemployment
#####################################

# Calculate the mean of UNEMPLOYMENT
mean_UNEMPLOYMENT <- mean(final_df$UNEMPLOYMENT, na.rm = TRUE)

# Calculate the difference from the mean for UNEMPLOYMENT
final_df$UNEMPLOYMENT_diff_from_mean <- final_df$UNEMPLOYMENT -
mean_UNEMPLOYMENT
```

Regions

Because we know that sales vary by region, we will perform one-hot encoding on the region variables to ensure they are represented in a format that is suitable for ML algorithms.

```r
###############################
# Feature Engineering - Regions
###############################

# Perform one-hot encoding for the 'Region' variable
# Convert categorical variables into a binary matrix
one_hot_region <- model.matrix(~ Region - 1, data = final_df)

# Combine the one-hot encoded matrix with the original dataframe
final_df <- cbind(final_df, one_hot_region)
```

Paydays

We believe that paydays may influence when and how often customers eat meals away from home. To validate this in our models, we feature engineer paydays to determine the number of days until the next payday.

```r
###############################
# Feature Engineering - PAYDAYS
###############################

# Initialize Days_until_next_payday to 0
final_df$Days_until_next_payday <- 0

# Find indices where PAY equals 1
payday_indices <- which(final_df$PAY == 1)

# Loop through each observation
for (i in 1:length(final_df$PAY)) {
  # Find the next payday index
  next_payday_index <- payday_indices[which(payday_indices > i)[1]]

  # Calculate the number of days until the next payday
```

```r
  if (!is.na(next_payday_index)) {
    final_df$Days_until_next_payday[i] <- next_payday_index - i
  }
}
```

Holidays

We believe that holidays may influence consumer visitation patterns, both positively and negatively. We also believe that the days leading up to, and the days after a holiday may effect visitation rates. Therefore we feature engineer holidays by denoting the holiday itself with a binary variable, and we do the same for days before and days after a holiday.

```r
################################
# Feature Engineering - Holidays
################################

# Initialize variables for days before and after a holiday
final_df$Before_holiday <- 0
final_df$After_holiday <- 0

# Find indices where HOLIDAY equals 1
holiday_indices <- which(final_df$HOLIDAY == 1)

# Create indicator variables for days before and after a holiday
for (i in holiday_indices) {
  final_df$Before_holiday[(i-3):(i-1)] <- 1  # Set values to 1 for three days
before the holiday
  final_df$After_holiday[(i+1):(i+3)] <- 1   # Set values to 1 for three days
after the holiday
}
```

Here is the updated structure of our dataframe after feature engineering is complete.

```
## 'data.frame':    80759 obs. of  34 variables:
##  $ Store                        : int  36530 36530 36530 36530 36530 36530 .
##  $ Date                         : Date, format: "2019-01-01" "2019-01-02" ..
##  $ Sales                        : num  4978 6052 7130 7021 7479 ...
##  $ TAVG                         : num  36 31.5 35 39.5 47.5 45 57 46.5 32.5
##  $ GAS                          : num  2.06 2.06 2.06 2.06 2.06 ...
##  $ CPI                          : num  234 234 234 234 234 ...
##  $ UNEMPLOYMENT                 : num  3.5 3.5 3.5 3.5 3.5 3.5 3.5 3.5 3.5 .
##  $ HOLIDAY                      : int  1 0 0 0 0 0 0 0 0 0 ...
##  $ PAY                          : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ MON                          : int  0 0 0 0 0 0 1 0 0 0 ...
##  $ TUE                          : int  1 0 0 0 0 0 0 1 0 0 ...
##  $ WED                          : int  0 1 0 0 0 0 0 0 1 0 ...
##  $ THU                          : int  0 0 1 0 0 0 0 0 0 1 ...
##  $ FRI                          : int  0 0 0 1 0 0 0 0 0 0 ...
##  $ SAT                          : int  0 0 0 0 1 0 0 0 0 0 ...
##  $ SUN                          : int  0 0 0 0 0 1 0 0 0 0 ...
```

```
##  $ Region                         : chr  "Midwest" "Midwest" "Midwest" st" ...
##  $ City                           : chr  "Hannibal" "Hannibal" "Hannibal"  ...
##  $ State                          : chr  "MO" "MO" "MO" "MO" ...
##  $ Zip.Code                       : int  63401 63401 63401 63401 63401 63401 .
##  $ Month                          : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ Day                            : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Quarter                        : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ DayOfWeek                      : num  3 4 5 6 7 1 2 3 4 5 ...
##  $ TAVG_diff_from_mean            : num  -15.92 -20.42 -16.92 -12.42 -4.42 ...
##  $ CPI_diff_from_mean             : num  -54.5 -54.5 -54.5 -54.5 -54.5 ...
##  $ UNEMPLOYMENT_diff_from_mean: num  -0.96 -0.96 -0.96 -0.96 -0.96 ...
##  $ RegionColorado Springs     : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ RegionMidwest                  : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ RegionNew Mexico               : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ RegionUpstate NY               : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Days_until_next_payday     : num  30 29 28 27 26 25 24 23 22 21 ...
##  $ Before_holiday                 : num  0 0 1 1 1 1 1 1 1 1 ...
##  $ After_holiday                  : num  0 1 1 1 0 0 0 0 0 0 ...
```

Training and Testing Sets

Now that we have explored the data and feature engineered the data, it is time to create training and testing sets for the models. We split our training and testing data by year, with the training data including the years 2019, 2020, 2021 and 2022; that is 80% of the data, and the testing data being 2023; that is 20% of the data.

```r
# Create train and test sets

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

# Define split date (end of 2022)
split_date <- as.Date("2022-12-31")

# Split data into training and test sets
training_data <- filter(final_df, Date <= split_date)
testing_data <- filter(final_df, Date > split_date)
```

Results

This section contains the results of all models before the ensemble stacking process.

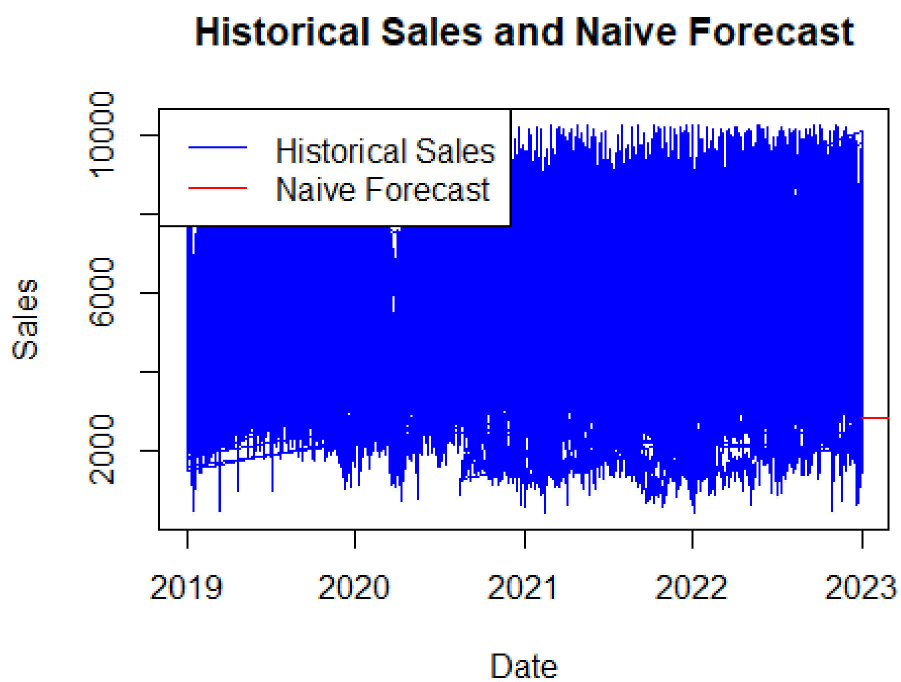
Traditional Time Series Models

Traditional time series models like Linear, ARIMA, and STLF are widely used in forecasting because they offer a structured framework for capturing various components of time series data and making predictions based on historical patterns.

We also choose to include the Prophet model because it is adept at handling time series data with irregularities, while also accommodating multiple seasonality and holiday effects, which we know exist in our data.

Before we begin with these models, we create a naive baseline forecast that assumes the future values will be equal to the most recent value. This establishes a baseline for comparison with more sophisticated forecasting methods.

Naive Forecast



Summary of Models

Evaluation_Metric	Naive
RMSE	3217.61010
MAPE	44.98366
Forecast_Bias	2650.00927
Forecast_Accuracy	55.91194

The results of the naive forecast suggests that the forecasted values are over 3,000 units away from the actual values, the forecasted values deviate from the actual values by 45%, the forecast tends to overestimate the values by over 2,600 units, and approximately 56% of the forecasted values are accurate compared to the actual values.

As a baseline forecast for comparison, these evaluation metrics suggest that there is room for improvement in our predictions.

Linear Regression Model

A linear regression model is used to analyze the relationship between a dependent variable and independent variables by fitting a linear equation to the observed data.

Traditionally, the company has used a linear regression model that includes a dataset of dates, stores, sales, seasonality, holidays, and paydays. Here are the evaluation metrics of the original company model.

Summary of Models - Original Company Model

Evaluation_Metric	Linear
RMSE	1646.51404
MAPE	31.85637
Forecast Bias	0.00000
Forecast Accuracy	78.21815

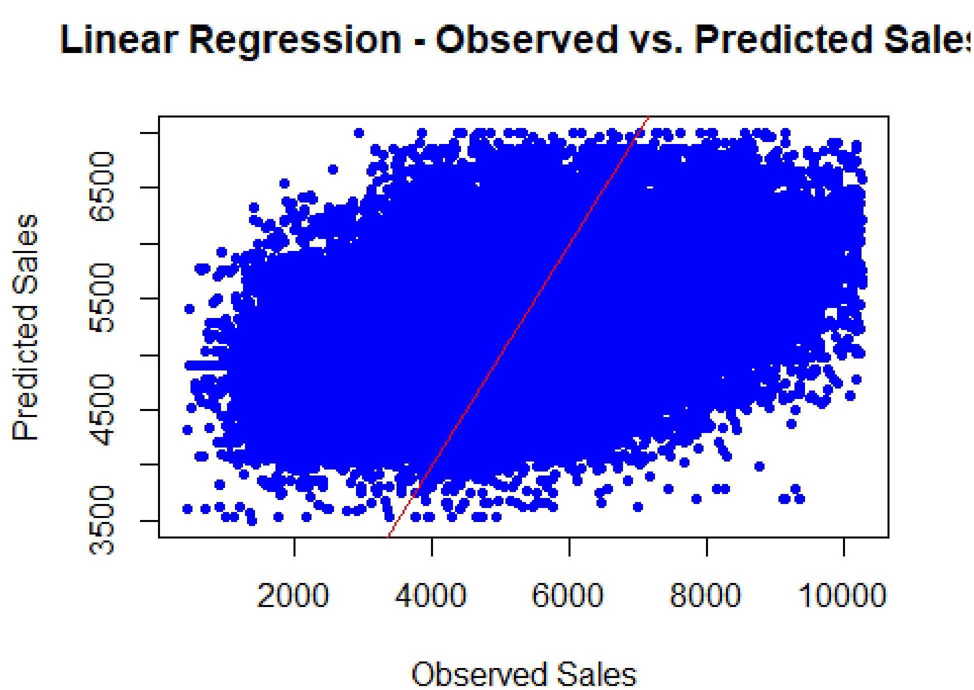
The results of the company's original LR model suggest that the forecasted values are over 1,600 units away from the actual values, the forecasted values deviate from the actual values by 32%, the forecast is neither overestimating or underestimating the actual values, and approximately 78% of the forecasted values are accurate compared to the actual values.

This original LR model is obviously much better than the naive forecast. Next we seek to improve upon the original LR model with the addition of third-party data like temperatures, gas prices, Consumer Price Index, and unemployment rate. Here are the results of the updated linear model with the third-party data added.

```
##
## Call:
## lm(formula = Sales ~ MON + TUE + WED + THU + FRI + SAT + SUN +
##     GAS + HOLIDAY + PAY + TAVG_diff_from_mean + CPI_diff_from_mean +
##     UNEMPLOYMENT_diff_from_mean + Days_until_next_payday + +Before_holiday
+
##     After_holiday, data = training_data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5155.7 -1181.9   -57.5  1133.6  5666.9
##
## Coefficients: (1 not defined because of singularities)
##                              Estimate Std. Error t value Pr(>|t|)
## (Intercept)                 6063.8768  1149.7888   5.274 1.34e-07 ***
## MON                          153.8426    24.3631   6.315 2.73e-10 ***
## TUE                          322.1449    24.5457  13.124  < 2e-16 ***
## WED                          424.9228    24.5242  17.327  < 2e-16 ***
## THU                          659.4286    24.5351  26.877  < 2e-16 ***
## FRI                         1125.8121    24.6291  45.711  < 2e-16 ***
## SAT                          598.5543    24.4129  24.518  < 2e-16 ***
```

```
## SUN                                    NA          NA        NA        NA
## GAS                              106.2769     15.6427     6.794 1.10e-11 ***
## HOLIDAY                         -375.8676     30.3260   -12.394  < 2e-16 ***
## PAY                              189.4575     38.6051     4.908 9.24e-07 ***
## TAVG_diff_from_mean                6.7097      0.4064    16.511  < 2e-16 ***
## CPI_diff_from_mean                15.6462      0.4839    32.331  < 2e-16 ***
## UNEMPLOYMENT_diff_from_mean       25.9593      3.3848     7.669 1.75e-14 ***
## Days_until_next_payday             2.9739      0.7686     3.869 0.000109 ***
## Before_holiday                 -1446.5426   1148.2014    -1.260 0.207734
## After_holiday                    -81.2391     19.3007    -4.209 2.57e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1623 on 62487 degrees of freedom
## Multiple R-squared:  0.09946,    Adjusted R-squared:  0.09925
## F-statistic: 460.1 on 15 and 62487 DF,  p-value: < 2.2e-16
```

We see a NA for Sun(day), indicating a singularity error. Upon examination, the data structure of this variable is correct. This variable may not provide much unique information or predictive power for forecasting, leading to numerical instability in the regression coefficients. Or the regression model may be overfitting the data, leading to numerical issues during estimation. We move forward without Sunday as a predictive variable.



Summary of Models

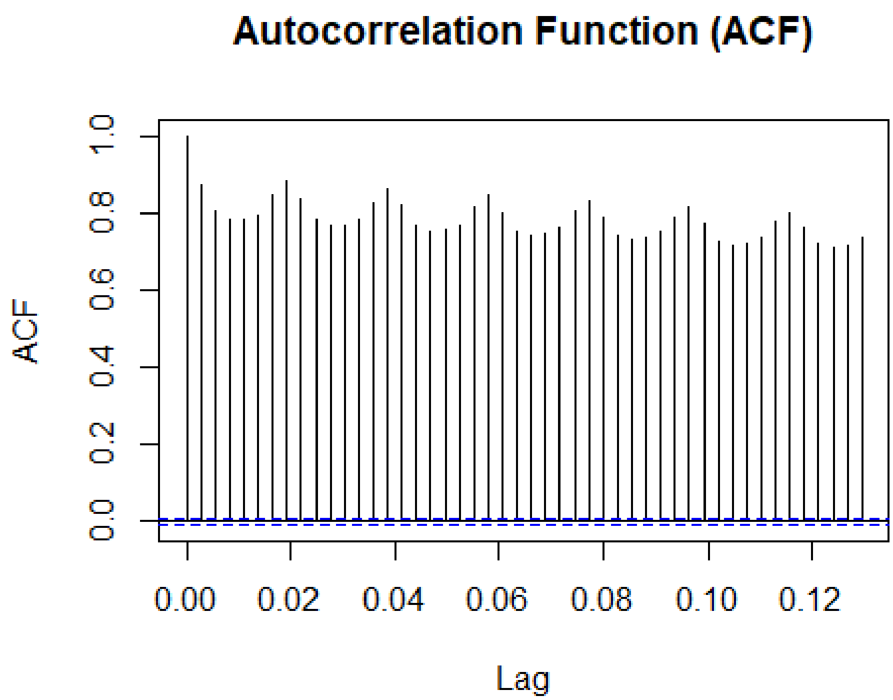
Evaluation_Metric	Naive	New_Linear
RMSE	3217.61010	1623.11461
MAPE	44.98366	31.32334
Forecast Bias	2650.00927	0.00000
Forecast Accuracy	55.91194	78.32954

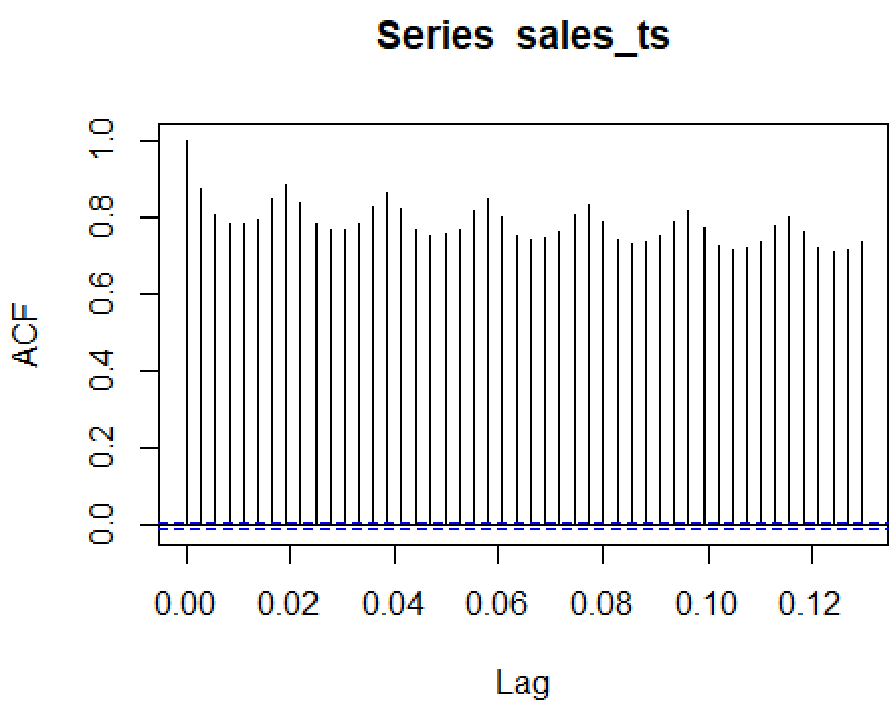
When third-party data was added to the LR model, no improvement was seen in the evaluation metrics relative to the original LR model that the company is currently using.

ARIMA Model

The ARIMA model is a widely used time series forecasting technique that combines autoregression, differencing, and moving average components.

First, we create an Autocorrelation Function (ACF) plot to visualize the correlation between the time series sales data and its lagged values.





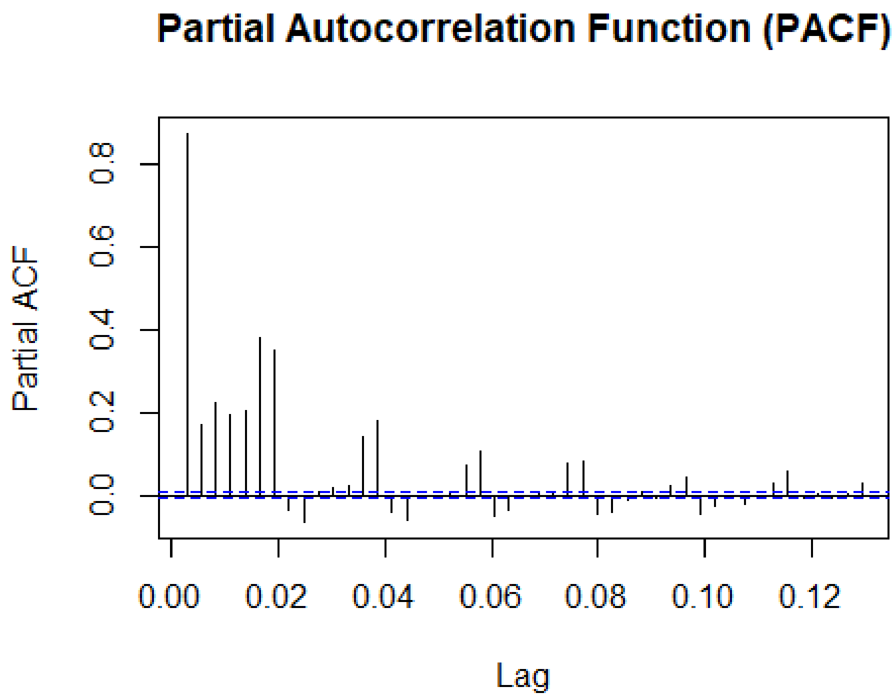

```
##    Lag       ACF
## 1    0 1.0000000
## 2    1 0.8746515
## 3    2 0.8052827
```

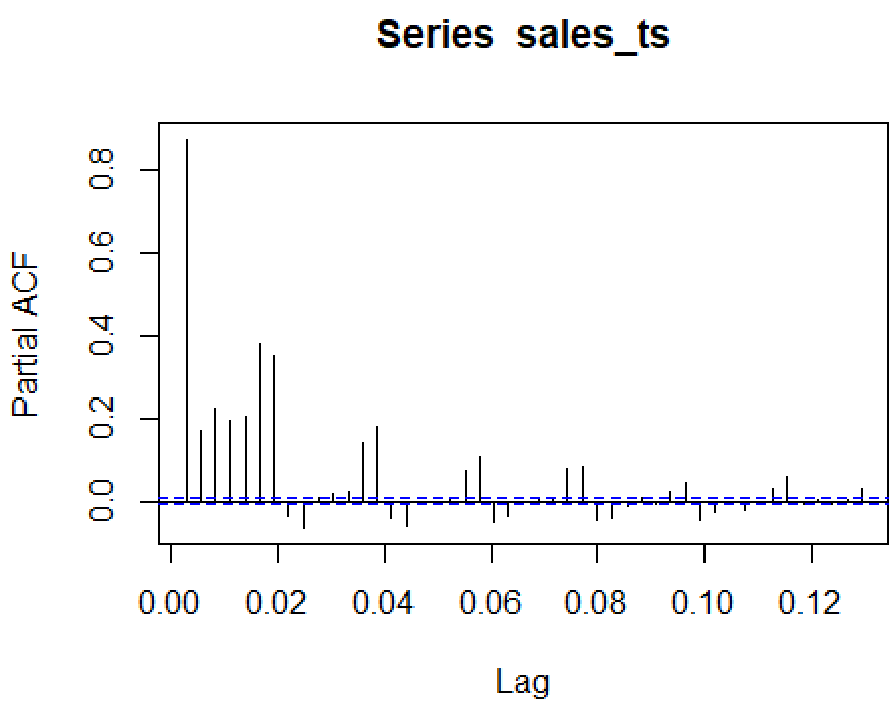
```
## 4    3 0.7840655
## 5    4 0.7829698
## 6    5 0.7944614
## 7    6 0.8455499
## 8    7 0.8845587
## 9    8 0.8362107
## 10   9 0.7836293
## 11  10 0.7681320
## 12  11 0.7694563
## 13  12 0.7820650
## 14  13 0.8285908
## 15  14 0.8632557
## 16  15 0.8195843
## 17  16 0.7687357
## 18  17 0.7547348
## 19  18 0.7564795
## 20  19 0.7706033
## 21  20 0.8142222
## 22  21 0.8450081
## 23  22 0.8016289
## 24  23 0.7546512
## 25  24 0.7425959
## 26  25 0.7463005
## 27  26 0.7611557
## 28  27 0.8061488
## 29  28 0.8330988
## 30  29 0.7892544
## 31  30 0.7418589
## 32  31 0.7307463
## 33  32 0.7374007
## 34  33 0.7517043
## 35  34 0.7913082
## 36  35 0.8145409
## 37  36 0.7712705
## 38  37 0.7268406
## 39  38 0.7185291
## 40  39 0.7225526
## 41  40 0.7378903
## 42  41 0.7771919
## 43  42 0.8009861
## 44  43 0.7609695
## 45  44 0.7201193
## 46  45 0.7098905
## 47  46 0.7162324
## 48  47 0.7347902
```

Our interpretation of the plot is that the dotted lines (the significance thresholds) are close to zero, suggesting no linear relationship between observations at the corresponding lags. The black bars (the autocorrelation coefficients) are relatively constant with a slight decay as the lag increases, suggesting the presence of a moving average in the time series data.

The ACF results indicate an autocorrelation structure in the time series data with high autocorrelation coefficients persisting over multiple lags.

A Partial Autocorellation plot (PACF) is another graphical tool to visualize the partial correlation between the time series sales data and its lagged values.



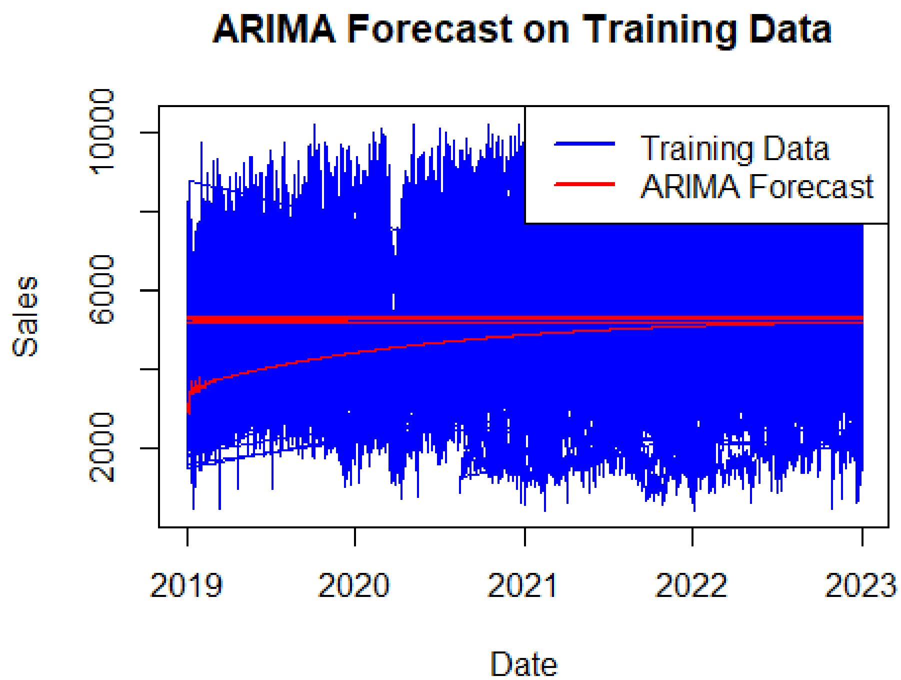


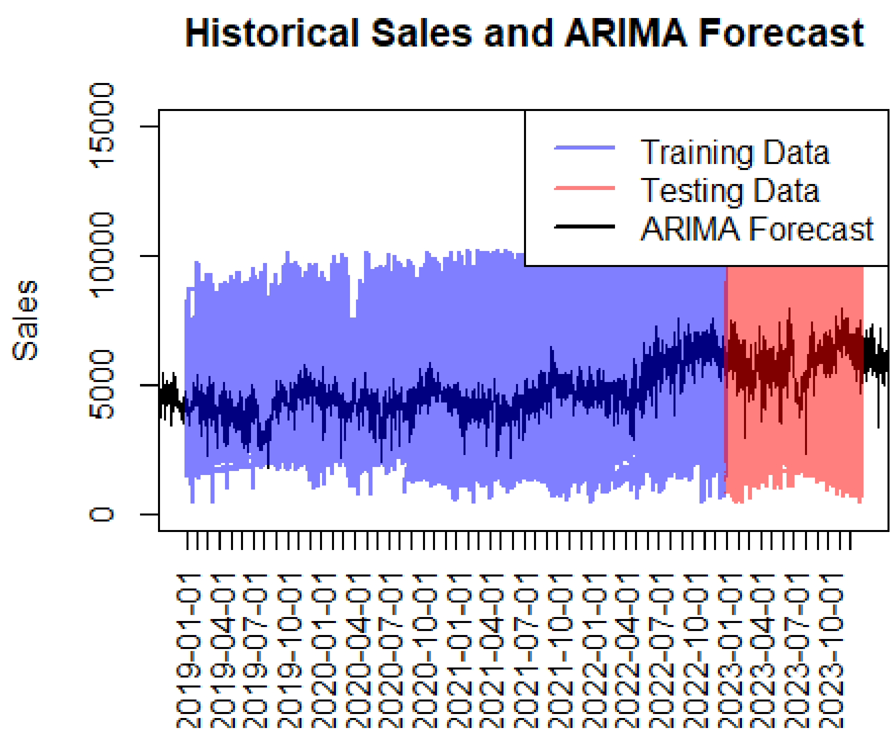
```
##    Lag          PACF
## 1    0  8.746515e-01
## 2    1  1.713618e-01
## 3    2  2.215809e-01
## 4    3  1.936408e-01
## 5    4  2.050228e-01
## 6    5  3.797966e-01
## 7    6  3.497383e-01
## 8    7 -3.740811e-02
## 9    8 -6.440688e-02
## 10   9  7.622432e-03
## 11  10  1.817774e-02
## 12  11  2.239159e-02
## 13  12  1.385021e-01
## 14  13  1.774913e-01
## 15  14 -4.028416e-02
## 16  15 -6.101067e-02
## 17  16 -6.667918e-03
## 18  17 -5.095907e-03
## 19  18  6.331316e-03
## 20  19  7.334603e-02
## 21  20  1.072356e-01
## 22  21 -5.010535e-02
## 23  22 -3.732781e-02
## 24  23 -1.814978e-03
## 25  24  2.017916e-03
## 26  25  6.457613e-03
## 27  26  7.594653e-02
## 28  27  8.101003e-02
## 29  28 -4.315235e-02
## 30  29 -4.238621e-02
## 31  30 -8.794118e-03
## 32  31  9.984529e-03
## 33  32 -8.117384e-03
## 34  33  2.209124e-02
## 35  34  4.320249e-02
## 36  35 -4.519764e-02
## 37  36 -2.727980e-02
## 38  37  3.718907e-05
## 39  38 -2.102827e-02
## 40  39 -5.634827e-03
## 41  40  2.588706e-02
## 42  41  5.930033e-02
## 43  42 -7.311326e-03
## 44  43  5.692828e-03
## 45  44 -4.988788e-03
## 46  45  5.466467e-03
## 47  46  2.600568e-02
```

We notice that as the lag increases, PACF (the dotted horizontal line) remains around zero, indicating neither a positive or negative correlation. Some coefficients (the black bars) become non-significant, close to zero, at certain lags, indicating a weak association between the observations. In general we are seeing weak positive and some negative autocorrelations for the lags. This indicates that there are some residual patterns or dependencies in the time series data that are not explained by the immediate preceding observations. These residual patterns could be due to factors such as seasonality, trend, or other underlying dynamics.

The ARIMA model contains tuning components including autoregressive, integrated, moving average, and seasonal components. Each of the parameters below were adjusted and tuned, and evaluation metrics were run on the model after each tuning until we produced a model with the best evaluation metrics. The final tuned parameters are:

```
# Autoregressive Component (p)
# Integrated Component (d)
# Moving Average Component (q)
# Seasonal Autoregressive Component (P)
# Seasonal Integrated Component (D)
# Seasonal Moving Average Component (Q)
# Seasonal Frequency (m)
p <- 1
d <- 0
q <- 1
P <- 1
D <- 0
Q <- 1
m <- 30 #a seasonal pattern that repeats every 30 days
```


ARIMA Forecast on Training Data
Training Data
ARIMA Forecast
Sales
2000
6000
10000
2019
2020
2021
2022
2023
Date


Historical Sales and ARIMA Forecast
Training Data
Testing Data
ARIMA Forecast
Sales
0
5000
10000
15000
2019-01-01
2019-04-01
2019-07-01
2019-10-01
2020-01-01
2020-04-01
2020-07-01
2020-10-01
2021-01-01
2021-04-01
2021-07-01
2021-10-01
2022-01-01
2022-04-01
2022-07-01
2022-10-01
2023-01-01
2023-04-01
2023-07-01
2023-10-01

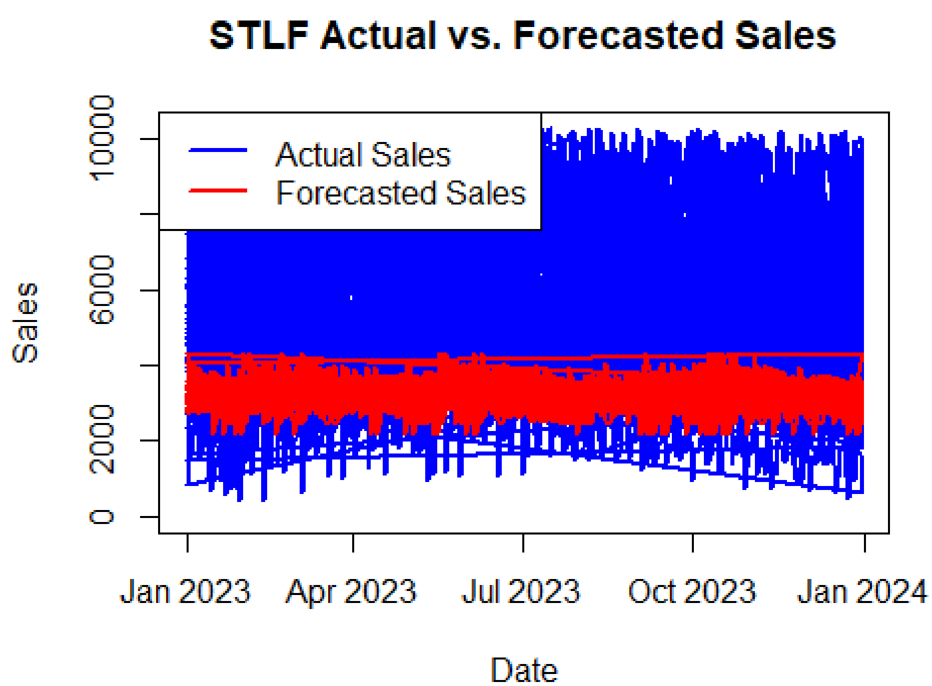
Summary of Models

Evaluation_Metric	Naive	Linear	ARIMA
RMSE	3217.61010	1623.11461	1862.36789
MAPE	44.98366	31.32334	33.25723
Forecast_Bias	2650.00927	0.00000	212.04841
Forecast_Accuracy	55.91194	78.32954	76.23366

The results of the ARIMA model suggest that the forecasted values are over 1,800 units away from the actual values, the forecasted values deviate from the actual values by about 33%, the forecast is overestimating the actual values by about 212, and approximately 76% of the forecasted values are accurate compared to the actual values. We interpret these results to be better than the naive forecast, and similar, albeit slightly less effective, than the LR model.

STLF Model

Seasonal-Trend Decomposition using Loess (STL) is a technique used for decomposing a time series into three components: seasonal, trend, and residual. The STLF (Seasonal and Trend decomposition using Loess Forecasting) function in R is used to fit a forecasting model that incorporates STL to decompose the time series. After decomposition, it applies a forecasting method to predict future values.



Summary of Models

Evaluation_Metric	Naive	Linear	ARIMA	STLF
RMSE	3217.61010	1623.11461	1862.36789	2906.08221
MAPE	44.98366	31.32334	33.25723	39.69940
Forecast Bias	2650.00927	0.00000	212.04841	2239.84914
Forecast Accuracy	55.91194	78.32954	76.23366	61.87877

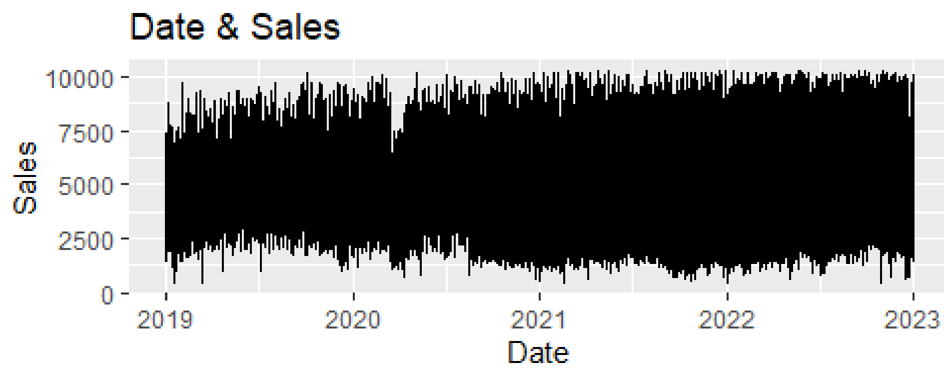
The results of the STLF model suggest that the forecasted values are nearly 3,000 units away from the actual values, the forecasted values deviate from the actual values by about 40%, the forecast is overestimating the actual values by over 2,000,and approximately 62% of the forecasted values are accurate compared to the actual values. These results are only slightly better than the Naive forecast, while being more biased and less accurate than the Linear and ARIMA models.

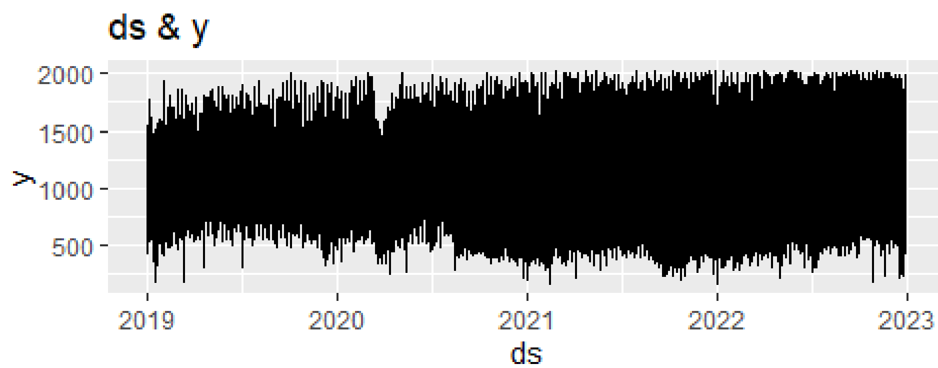
Prophet Model

Prophet is an open-source forecasting tool developed by Facebook, designed to handle time series data with seasonal effects and irregular trends. There are three main components of a Prophet model: trend, seasonality, and holiday effects. Prophet utilizes a flexible regression model to capture patterns and fluctuations in the data.

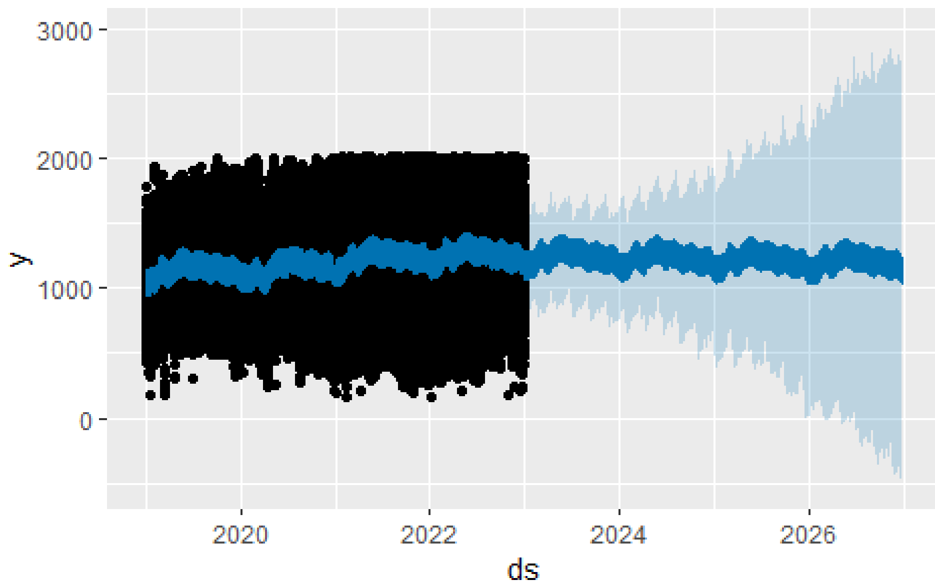
We will also conduct a Box-Cox transformation. Often in forecasting, we choose a specific type of power transformation to remove noise before feeding the data into a forecasting model. Box-Cox transforms are data transformations that evaluate a set of lambda coefficients and selects the value that achieves the best approximation of normality.

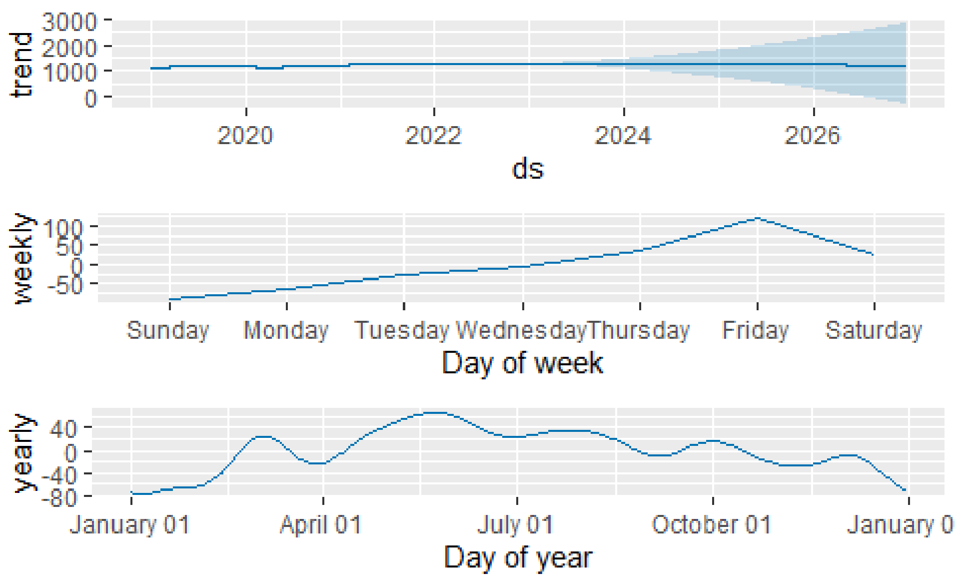
Prophet always expects two columns in the input data frame: ds and y, containing the date and numeric values respectively. The plots below confirms that our Box-Cox transform is working correctly as the bottom transformed plot matches the patterns of the original data in the top plot.



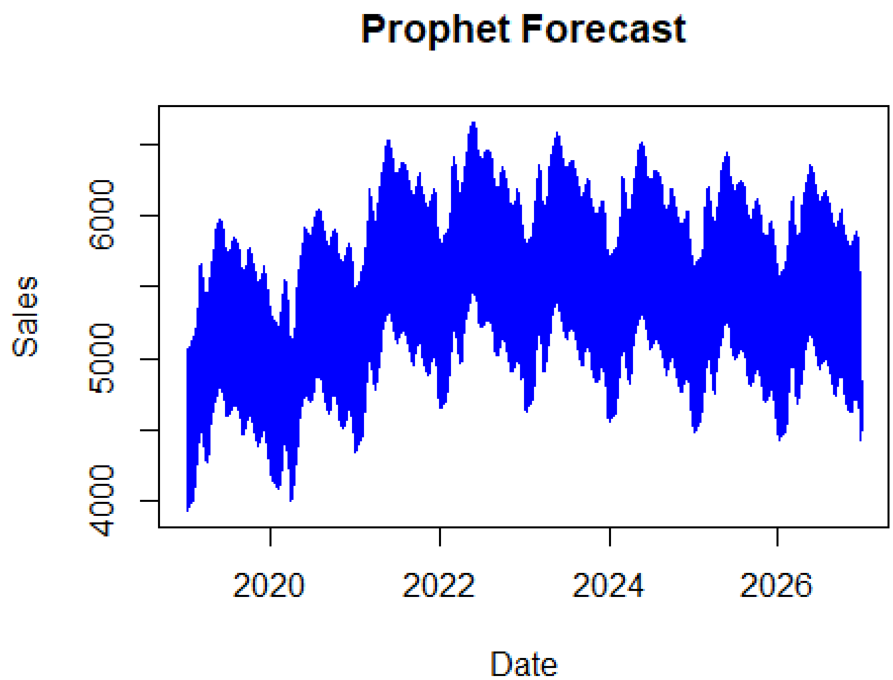


Below is a plot of the Prophet forecast, through the year 2026, along with a plot of the individual forecast components. The forecast and component visualizations show that Prophet was able to accurately model the underlying trend in the data, while also accurately modeling weekly and yearly seasonality.





Since the Prophet forecast was based on the Box-Cox transformed data, we need to transfer the forecasted values back to their original units. We do this by performing an inverse Box-Cox transformation. The Prophet forecast based on actual units is below.



Summary of Models

Evaluation_Metric	Naive	Linear	ARIMA	STLF	PROPHET
RMSE	3217.6101	1623.1146	1862.3678	2906.0822	1885.4627
MAPE	44.98366	31.32334	33.25723	39.69940	46.66521
Forecast Bias	2650.0092	0.00000	212.04841	2239.8491	-885.17100
Forecast Accuracy	55.91194	78.32954	76.23366	61.87877	71.81258

The results of the Prophet model suggest that the forecasted values are over 1,800 units away from the actual values, the forecasted values deviate from the actual values by about 47%, the forecast is underestimating the actual values by approximately -900, and approximately 72% of the forecasted values are accurate compared to the actual values. These results are nearer, albeit less effective than the linear and ARIMA models, and better than the STLF model.

Machine Learning Models

ML models offer advantages over traditional time series models, especially in sales forecasting scenarios. ML models can handle complex, nonlinear relationships and patterns in data. They have the capability to automatically learn and adapt to new data, enabling them to continuously improve over time. ML models can incorporate a wide range of predictive features beyond just historical sales data. They also offer scalability and flexibility to adapt to changing business environments. Overall, ML models should allow for more accurate predictions.

For our sales forecasting, we will run two ML models, the Random Forest (RF) model and the Gradient Boosting Machines (GBM) model.

Random Forest Model (RF)

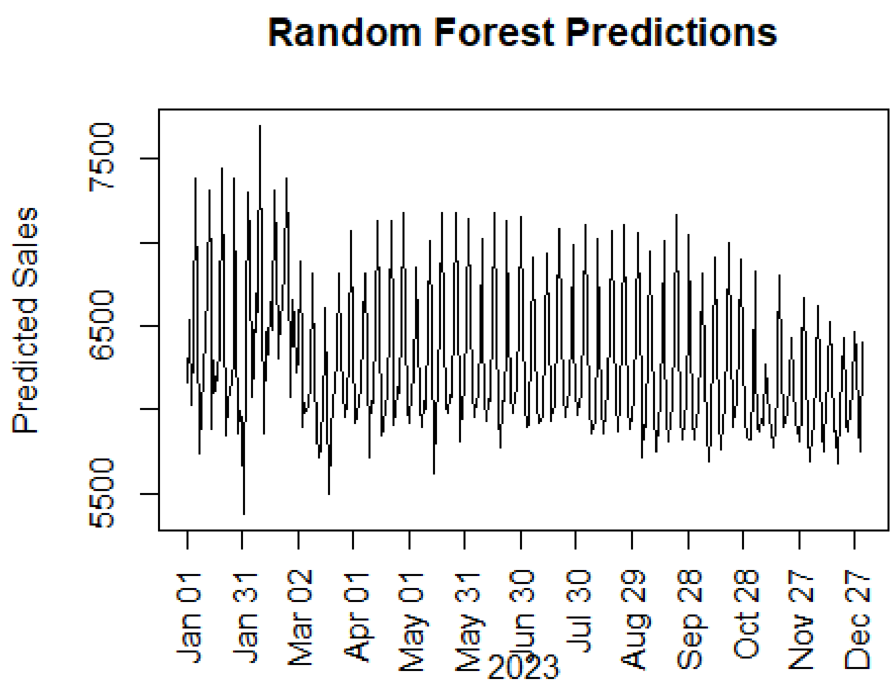
Random Forest is a ML algorithm that operates by constructing a multitude of decision trees during training and outputs the mean prediction of individual trees. RF models can handle large and complex datasets containing various features, the likes of which exist in our data. Its resistance to overfitting can contribute to more accurate sales predictions, and also provides insights into feature importance, helping us understand which factors drive sales the most.

We review the variable importance variables and determine that features like 'City', 'Zip.Code' and 'Gas' have very high importance scores, indicating that they significantly contribute to the model's predictions. Alternatively, features like 'Before_Holiday' and 'HOLIDAY' have relatively lower importance scores, suggesting they have less influence on the model's predictions.

```
##
## Call:
##  randomForest(formula = Sales ~ ., data = training_data[, c(predictors,
response)])
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 9
##
##           Mean of squared residuals: 1033685
##                     % Var explained: 64.67
```

```
##                              IncNodePurity
## TAVG                          4.106142e+09
## GAS                           7.363207e+09
## CPI                           6.755581e+09
## UNEMPLOYMENT                  2.366386e+09
## HOLIDAY                       8.934928e+08
## PAY                           1.520662e+08
## MON                           4.835079e+08
## TUE                           2.791065e+08
## WED                           3.200835e+08
## THU                           4.510449e+08
## FRI                           2.720570e+09
## SAT                           5.566462e+08
## SUN                           1.010961e+09
## Region                        2.054345e+09
## City                          6.106558e+10
## State                         5.595365e+09
## Zip.Code                      3.383405e+10
## Month                         2.337170e+09
## Day                           3.623051e+09
## Quarter                       7.004115e+08
```

```
## DayOfWeek                       5.722038e+09
## TAVG_diff_from_mean             4.128173e+09
## CPI_diff_from_mean              6.764990e+09
## UNEMPLOYMENT_diff_from_mean     2.357511e+09
## Days_until_next_payday          4.624646e+09
## Before_holiday                  9.319835e+04
## After_holiday                   4.798557e+08
```



Summary of Models

Evaluation_ Metric	Naive	Linear	ARIMA	STLF	PROPHET	RANDOM FOREST
RMSE	3217.610	1623.114	1862.367	2906.08	1885.462	1107.74769
MAPE	44.98366	31.32334	33.25723	39.6994	46.66521	19.75736
Forecast Bias	2650.009	0.00000	212.0484	2239.84	-885.1710	-190.72133
Forecast Accuracy	55.91194	78.32954	76.23366	61.8787	71.81258	21.86108

The results of the RF model suggest that the forecasted values are within 1,100 units of the actual values, the forecasted values deviate from the actual values by about 20%, the forecast is underestimating the actual values by ~200, and approximately 22% of the forecasted values are accurate compared to the actual values. While the Random Forest model appears to provide reasonably accurate predictions (as indicated by the RMSE and MAPE), there is room for overall performance improvement in terms of reducing forecast bias and increasing forecast accuracy.

Gradient Boosting Machines Model (GBM)

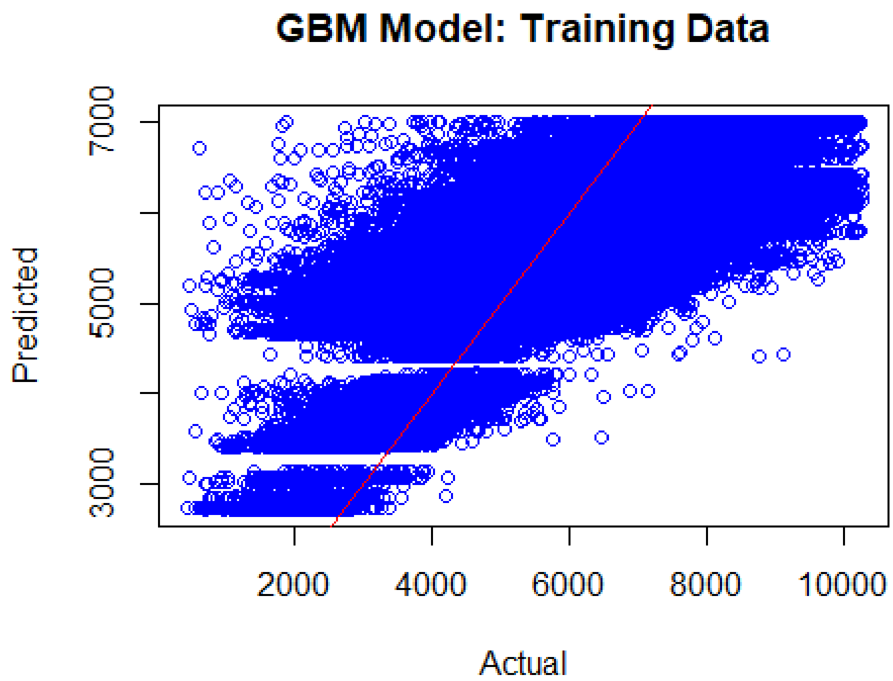
The Gradient Boosting Machines model is a ML technique that combines decision trees to create a predictive model. It works by sequentially adding new models to correct the errors made by previous models, reducing the overall prediction error. Each new model is trained on residuals of previous models, with the final prediction being the sum of all models.

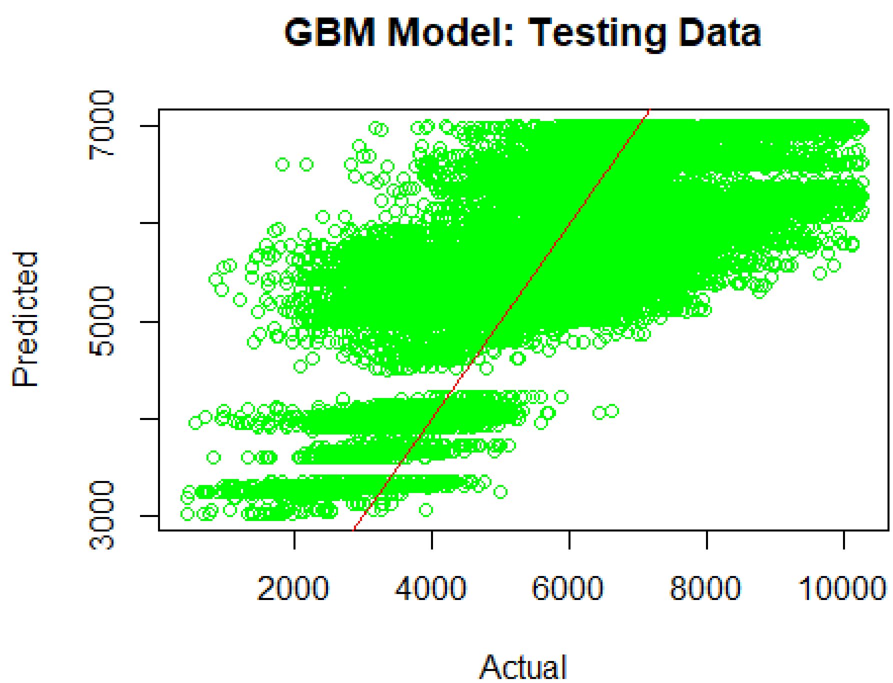
Both GBM and RF models use decision trees as their base, but they differ in how they build and combine them. RF models focus on reducing variance by averaging multiple independent models, while GBM models focus on reducing bias by emphasizing the correct prediction of instances that were previously misclassified. This often leads to GBM being more accurate but also potentially more prone to overfitting.

Our GBM model has been tuned by adjusting the various hyperparameters to find the combination that yields the best evaluation metrics. The final tuned parameters are:

```
##### Train the GBM model with training_data #####
  gbm_model <- gbm(
    formula = as.formula(paste(response, "~", paste(predictors, collapse = "
+ "))),
    data = training_data,
    distribution = "gaussian",
    n.trees = 200,
    interaction.depth = 5,
    shrinkage = 0.01,
    bag.fraction = 0.5,
    cv.folds = 5,  # Optional: perform cross-validation
    verbose = TRUE
  )

## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1  2901427.2924             nan     0.0100 24171.1245
##      2  2877512.6557             nan     0.0100 23549.6631
##      3  2853858.8944             nan     0.0100 23517.4754
##      4  2831023.5161             nan     0.0100 22695.7945
##      5  2808243.5448             nan     0.0100 22711.4487
##      6  2786634.4695             nan     0.0100 21573.1592
##      7  2765084.6291             nan     0.0100 21638.2379
##      8  2743796.8675             nan     0.0100 21120.7080
##      9  2722865.3174             nan     0.0100 20950.3418
##     10  2702648.7423             nan     0.0100 20324.7220
##     20  2516380.3436             nan     0.0100 17137.4519
##     40  2233686.9259             nan     0.0100 11871.2617
##     60  2032526.1418             nan     0.0100 8399.1660
##     80  1885377.8755             nan     0.0100 5998.4632
##    100  1776445.1778             nan     0.0100 4864.8483
##    120  1679507.1907             nan     0.0100 6340.3933
##    140  1607738.6569             nan     0.0100 3003.7742
##    160  1546490.3066             nan     0.0100 2684.9182
```

```
##     180   1501581.3743             nan     0.0100 1822.6693
##     200   1463923.9006             nan     0.0100 1390.9632
```





Summary of Models

Evaluation_ Metric	Naive	Linear	ARIMA	STLF	PROPHET	RANDOM_ FOREST	GBM
RMSE	3217.61	1623.11	1862.36	2906.	1885.462	1107.7476	1336.88
MAPE	44.9836	31.3233	33.2572	39.69	46.66521	19.75736	22.6558
Forecast Bias	2650.00	0.00000	212.048	2239.	-885.1710	-190.7213	0.11525
Forecast Accuracy	55.9119	78.3295	76.2336	61.87	71.81258	21.86108	16.8109

The results of the GBM model suggest that the forecasted values are approximately 1,300 units away from the actual values, the forecasted values deviate from the actual values by about 23%, the forecast is neither underestimating or overestimating the actual values, and approximately 17% of the forecasted values are accurate compared to the actual values. Similar to the RF model, the GBM model appears to provide somewhat accurate predictions (based on the RMSE and MAPE, and no bias), however their is room for improvement to better capture the underlying patterns and variability in the data.

Ensemble Stacking Models

In sales forecasting, ensemble stacking is a technique where multiple predictive models are combined or "stacked" together to improve the accuracy and the robustness of sales forecasts. This is a six step process.

Step 1. Model Selection

Based on the results of the models, we have chosen to stack the Linear, ARIMA, and GBM models. These will be classified as the base models.

Step 2. Train base models

We have already trained each of the selected models on historical sales data. In addition we have also used cross-validation techniques to tune hyperparameters to optimize the base model performance.

Step 3. Generate predictions

We have already made predictions on the training data using each base model. These predictions will serve as the input for the meta-model. This ensures the meta-model learns from the base model's predictions before applying it to unseen data.

Step 4. Prepare meta-model

The meta model, or "stacking model", is now trained using the predictions generated by the base model as features. The meta-model will learn how to combine or weigh the predictions from the base models to produce a final ensemble prediction.

```
head(meta_input)
```

```
##         Date true_target Prediction_Model1 Prediction_Model2
Prediction_Model3
## 1 2019-01-01     4978.24          5333.116          3068.798
6093.201
## 2 2019-01-02     6052.23          5697.354          3137.705
6124.095
## 3 2019-01-03     7130.21          4505.828          3115.940
6455.822
## 4 2019-01-04     7021.39          4999.431          2939.132
6684.821
## 5 2019-01-05     7478.59          4604.116          3198.735
6522.571
## 6 2019-01-06     5556.86          3985.814          3099.968
6054.294
```

Step 4a. Feature engineering on meta-model

While optional, we decide to conduct feature engineering specifically for the meta model input.

```
########################################
# 4a.  FEATURE ENGINEERING ON META_INPUT
########################################

##################
# RANKING FEATURES
##################
#Using ranking features helps meta-model identify which models consistently
perform better or worse across different instances.

# Compute ranking features for each base model
rank_model1 <- rank(meta_input$Prediction_Model1)
rank_model2 <- rank(meta_input$Prediction_Model2)
rank_model3 <- rank(meta_input$Prediction_Model3)

# Create a dataframe with ranking features
ranking_df <- data.frame(
  "Rank_Model1" = rank_model1,
  "Rank_Model2" = rank_model2,
  "Rank_Model3" = rank_model3
)

# Bind ranking features to the meta_input dataset
meta_input <- cbind(meta_input, ranking_df)

#####################
# DIFFERENCE FEATURES
#####################
# The absolute differences or percent differences between pairs of
predictions from the base models.
```

```r
# This can help meta-model correct for bias or errors in individual
predictions.

# Compute difference features between pairs of predictions
difference_model1_model2 <- abs(meta_input$Prediction_Model1 -
meta_input$Prediction_Model2)
difference_model1_model3 <- abs(meta_input$Prediction_Model1 -
meta_input$Prediction_Model3)
difference_model2_model3 <- abs(meta_input$Prediction_Model2 -
meta_input$Prediction_Model3)

# Create a dataframe with difference features
difference_df <- data.frame(
  "Difference_Model1_Model2" = difference_model1_model2,
  "Difference_Model1_Model3" = difference_model1_model3,
  "Difference_Model2_Model3" = difference_model2_model3
)

# Bind difference features to the meta_input dataset
meta_input <- cbind(meta_input, difference_df)

#####################
# GEOMETRIC MEAN
#####################
# The geometric mean tends to give less weight to extreme values and can help
mitigate the influence of outliers.

# Compute geometric mean of predictions for each row
geometric_mean_predictions <- apply(meta_input[, c("Prediction_Model1",
"Prediction_Model2", "Prediction_Model3")], 1, function(row) {
  exp(mean(log(row), na.rm = TRUE))
})

# Add geometric mean as a new feature to the meta_input dataset
meta_input$Geometric_Mean_Predictions <- geometric_mean_predictions

####################
# STANDARD DEVIATION
####################
# Compute standard deviation of predictions for each row
standard_deviation_predictions <- apply(meta_input[, c("Prediction_Model1",
"Prediction_Model2", "Prediction_Model3")], 1, sd)

# Add standard deviation as a new feature to the meta_input dataset
meta_input$Standard_Deviation_Predictions <- standard_deviation_predictions

########
# MEDIAN
########
```

```r
# Compute median of predictions for each row
median_predictions <- apply(meta_input[, c("Prediction_Model1",
"Prediction_Model2", "Prediction_Model3")], 1, median, na.rm = TRUE)

# Add median as a new feature to the meta_input dataset
meta_input$Median_Predictions <- median_predictions

#####################
# TIME-BASED FEATURES
#####################
# Extract time-based features
meta_input$HOLIDAY <- as.numeric(training_data$HOLIDAY)
meta_input$PAY <- as.numeric(training_data$PAY)
meta_input$MON <- as.numeric(weekdays(training_data$Date) == "Monday")
meta_input$TUE <- as.numeric(weekdays(training_data$Date) == "Tuesday")
meta_input$WED <- as.numeric(weekdays(training_data$Date) == "Wednesday")
meta_input$THU <- as.numeric(weekdays(training_data$Date) == "Thursday")
meta_input$FRI <- as.numeric(weekdays(training_data$Date) == "Friday")
meta_input$SAT <- as.numeric(weekdays(training_data$Date) == "Saturday")
meta_input$SUN <- as.numeric(weekdays(training_data$Date) == "Sunday")
meta_input$Month <- month(training_data$Date)
meta_input$Day <- day(training_data$Date)
meta_input$Quarter <- quarter(training_data$Date)
meta_input$DayOfWeek <- wday(training_data$Date)
meta_input$DayOfWeek <- as.numeric(meta_input$DayOfWeek)
meta_input$Days_until_next_payday <- training_data$Days_until_next_payday
meta_input$Before_holiday <- training_data$Before_holiday
meta_input$After_holiday <- training_data$After_holiday

# View the updated meta_input dataset, with feature engineering added
head(meta_input)

##          Date true_target Prediction_Model1 Prediction_Model2
Prediction_Model3
## 1 2019-01-01     4978.24          5333.116          3068.798
6093.201
## 2 2019-01-02     6052.23          5697.354          3137.705
6124.095
## 3 2019-01-03     7130.21          4505.828          3115.940
6455.822
## 4 2019-01-04     7021.39          4999.431          2939.132
6684.821
## 5 2019-01-05     7478.59          4604.116          3198.735
6522.571
## 6 2019-01-06     5556.86          3985.814          3099.968
6054.294
##   Rank_Model1 Rank_Model2 Rank_Model3 Difference_Model1_Model2
## 1       33249           4     50106.0                2264.3174
## 2       47357           7     50706.5                2559.6492
## 3        3590           6     56122.5                1389.8879
```

```
## 4       18002           3     59212.0                2060.2986
## 5        5461           9     56661.0                1405.3811
## 6         210           5     48993.5                 885.8459
##   Difference_Model1_Model3 Difference_Model2_Model3
Geometric_Mean_Predictions
## 1                 760.0850                 3024.402
4637.297
## 2                 426.7403                 2986.389
4783.830
## 3                1949.9941                 3339.882
4491.986
## 4                1685.3902                 3745.689
4613.989
## 5                1918.4549                 3323.836
4579.813
## 6                2068.4798                 2954.326
4213.528
##   Standard_Deviation_Predictions Median_Predictions HOLIDAY PAY MON TUE
WED THU
## 1                       1573.312           5333.116       1   0   0   1
0   0
## 2                       1615.159           5697.354       0   0   0   0
1   0
## 3                       1677.750           4505.828       0   0   0   0
0   1
## 4                       1875.969           4999.431       0   0   0   0
0   0
## 5                       1668.505           4604.116       0   0   0   0
0   0
## 6                       1516.101           3985.814       0   0   0   0
0   0
##   FRI SAT SUN Month Day Quarter DayOfWeek Days_until_next_payday
Before_holiday
## 1   0   0   0     1   1       1         3                     30
0
## 2   0   0   0     1   2       1         4                     29
0
## 3   0   0   0     1   3       1         5                     28
1
## 4   1   0   0     1   4       1         6                     27
1
## 5   0   1   0     1   5       1         7                     26
1
## 6   0   0   1     1   6       1         1                     25
1
##   After_holiday
## 1             0
## 2             1
## 3             1
## 4             1
```

```
## 5            0
## 6            0
```

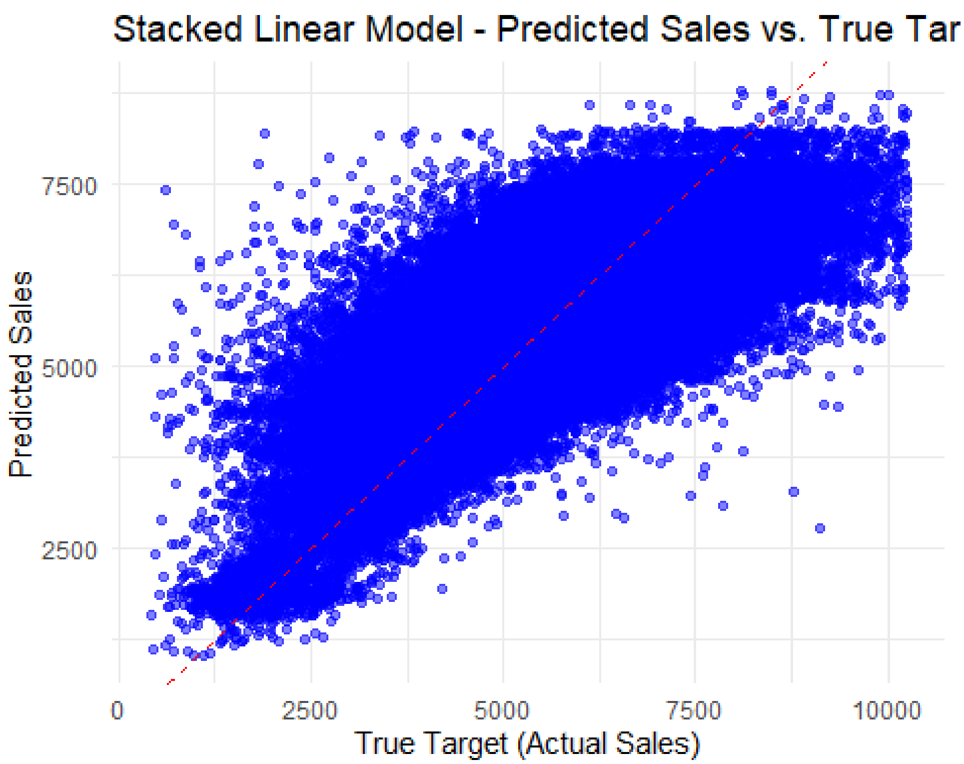
Step 5a. Generate predictions - Stacked Linear Regression

During the forecasting stage, each base model generated predictions. Now these predictions are fed into the meta model, and we are ready to produced a stacked prediction. We will start with the stacked linear regression model.

```
##
## Call:
## lm(formula = linear_formula, data = meta_input)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -6798.1  -698.9     1.2   696.2  6351.4
##
## Coefficients: (2 not defined because of singularities)
##                                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)                   -4.371e+03  8.870e+02  -4.928 8.32e-07 ***
## Date                           3.261e-01  1.919e-02  16.988  < 2e-16 ***
## Prediction_Model1             -2.456e+00  1.923e-01 -12.766  < 2e-16 ***
## Prediction_Model2             -3.230e+00  2.017e-01 -16.010  < 2e-16 ***
## Prediction_Model3             -1.522e+00  2.051e-01  -7.421 1.17e-13 ***
## Rank_Model1                   -6.673e-03  1.297e-03  -5.144 2.70e-07 ***
## Rank_Model2                    1.608e-02  4.134e-04  38.912  < 2e-16 ***
## Rank_Model3                    1.559e-02  1.419e-03  10.981  < 2e-16 ***
## Difference_Model1_Model2      -6.422e-03  4.832e-02  -0.133 0.894269
## Difference_Model1_Model3      -4.306e-03  5.565e-02  -0.077 0.938322
## Difference_Model2_Model3       1.789e-01  5.424e-02   3.298 0.000974 ***
## Geometric_Mean_Predictions     7.161e+00  5.507e-01  13.004  < 2e-16 ***
## Standard_Deviation_Predictions 3.006e-01  2.049e-01   1.467 0.142412
## Median_Predictions             2.369e-01  3.128e-02   7.572 3.73e-14 ***
## HOLIDAY                       -4.615e+02  2.320e+01 -19.892  < 2e-16 ***
## PAY                            1.470e+02  2.812e+01   5.226 1.74e-07 ***
## MON                            1.810e+02  1.725e+01  10.492  < 2e-16 ***
## TUE                            2.892e+02  1.863e+01  15.525  < 2e-16 ***
## WED                            3.316e+02  1.973e+01  16.806  < 2e-16 ***
## THU                            4.571e+02  2.280e+01  20.050  < 2e-16 ***
## FRI                            7.137e+02  3.055e+01  23.358  < 2e-16 ***
## SAT                            3.679e+02  2.183e+01  16.854  < 2e-16 ***
## SUN                                   NA         NA      NA       NA
## Month                          2.958e+01  5.480e+00   5.398 6.76e-08 ***
## Day                            9.460e+00  1.122e+00   8.433  < 2e-16 ***
## Quarter                       -6.012e+01  1.686e+01  -3.566 0.000362 ***
## DayOfWeek                             NA         NA      NA       NA
## Days_until_next_payday         1.214e+01  1.151e+00  10.544  < 2e-16 ***
## Before_holiday                 1.014e+03  7.988e+02   1.269 0.204417
## After_holiday                 -1.253e+02  1.357e+01  -9.235  < 2e-16 ***
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1118 on 62475 degrees of freedom
## Multiple R-squared:  0.5727, Adjusted R-squared:  0.5725
## F-statistic:  3101 on 27 and 62475 DF,  p-value: < 2.2e-16
```

We see NAs for Sun(day) and DayOfWeek, indicating singularity errors. Upon thorough examination, the data structure of these variables are correct. These variables may not provide much unique information or predictive power for forecasting, leading to numerical instability in the regression coefficients. Or the regression model may be overfitting the data, leading to numerical issues during estimation. We move forward with the stacked LR model without Sun and DayOfWeek as predictor variables.



Summary of Models

Evaluation_Metric	Stacked_Linear
RMSE	1118.10873
MAPE	18.72952
Forecast Bias	0.00000
Forecast Accuracy	85.07572

The results of the stacked linear model suggest that the forecasted values are approximately 1,100 units away from the actual values, the forecasted values deviate from the actual values by about 19%, the forecast is neither underestimating or overestimating the actual values, and approximately 85% of the forecasted values are accurate compared to the actual values. This stacked LR model is an improvement over the base models, and could serve as a future forecast model for the company.

Step 5b. Generate predictions - Stacked Gradient Boosting Machines

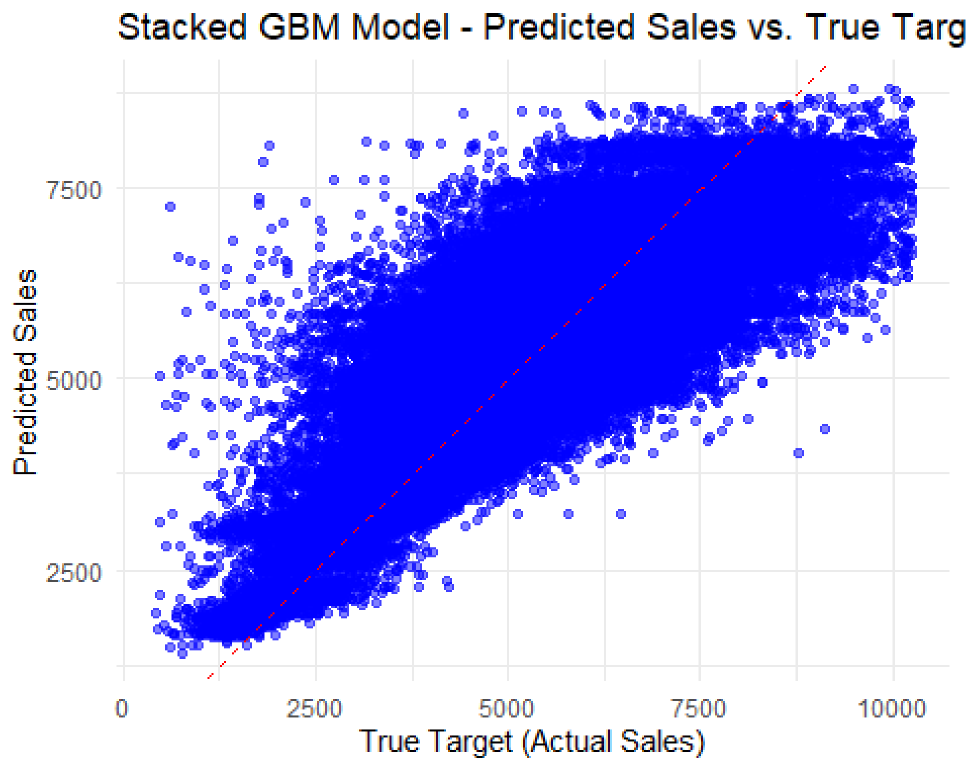
Now that the stacked linear regression model is complete, we will attempt a stacked GBM model with tuned hyperparameters.

```
##############################
# 4b.  TRAIN META-MODEL - GBM
##############################

set.seed(123)

# Define predictors (excluding true_target and Date if present)
predictors <- setdiff(names(meta_input), c("true_target", "Date"))

# Define response variable
response <- "true_target"

# Train the GBM model
stacked_gbm_model <- gbm(
  formula = as.formula(paste(response, "~", paste(predictors, collapse = " +
"))),
  data = meta_input,
  distribution = "gaussian",
  n.trees = 500,
  interaction.depth = 9,
  shrinkage = 0.01,
  bag.fraction = 0.9,
  cv.folds = 10,  # Optional: perform cross-validation
  verbose = TRUE
)

## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1  2891852.7584             nan     0.0100 33210.7949
##      2  2858825.4086             nan     0.0100 32537.8589
##      3  2826453.7101             nan     0.0100 31869.1284
##      4  2794438.1505             nan     0.0100 31014.8949
##      5  2763331.5140             nan     0.0100 32050.3257
##      6  2732854.1306             nan     0.0100 30817.3286
##      7  2702788.1560             nan     0.0100 30320.4570
##      8  2673248.8903             nan     0.0100 29161.8223
##      9  2644286.2397             nan     0.0100 28988.1568
##     10  2615825.9051             nan     0.0100 27982.2728
##     20  2361142.0540             nan     0.0100 23174.9922
##     40  1975341.2763             nan     0.0100 15167.0477
##     60  1709788.5127             nan     0.0100 10970.8734
##     80  1527047.4624             nan     0.0100 7362.4077
##    100  1400551.6019             nan     0.0100 4937.7339
```

```
##    120  1310597.7376            nan     0.0100 3727.5738
##    140  1247798.9687            nan     0.0100 2520.0737
##    160  1202667.2532            nan     0.0100 2024.7856
##    180  1171040.7467            nan     0.0100 1196.5825
##    200  1149432.4128            nan     0.0100 -1345.6723
##    220  1134111.8483            nan     0.0100  894.6611
##    240  1119127.3312            nan     0.0100  560.4633
##    260  1108252.1802            nan     0.0100  819.5599
##    280  1096924.7661            nan     0.0100  489.9934
##    300  1086187.7907            nan     0.0100  330.9558
##    320  1077615.4628            nan     0.0100  384.6984
##    340  1070884.5110            nan     0.0100  368.7130
##    360  1064770.4395            nan     0.0100  182.6391
##    380  1057207.5485            nan     0.0100  163.9808
##    400  1050538.5612            nan     0.0100  374.6576
##    420  1044409.6273            nan     0.0100  215.0740
##    440  1039316.6766            nan     0.0100  184.8767
##    460  1036778.8174            nan     0.0100  128.3443
##    480  1032486.8172            nan     0.0100  224.2260
##    500  1027911.8351            nan     0.0100 -140.7418
```

```
##                                                       var      rel.inf
## Prediction_Model3                       Prediction_Model3 6.825360e+01
## Difference_Model2_Model3         Difference_Model2_Model3 9.113043e+00
## Prediction_Model2                       Prediction_Model2 8.345715e+00
## Geometric_Mean_Predictions     Geometric_Mean_Predictions 6.705229e+00
## Median_Predictions                     Median_Predictions 2.377547e+00
## Prediction_Model1                       Prediction_Model1 1.427172e+00
## Standard_Deviation_Predictions Standard_Deviation_Predictions 8.725295e-01
## Month                                               Month 6.015860e-01
## Days_until_next_payday             Days_until_next_payday 5.772438e-01
## Difference_Model1_Model2         Difference_Model1_Model2 4.750931e-01
## Difference_Model1_Model3         Difference_Model1_Model3 3.784705e-01
## FRI                                                   FRI 2.396447e-01
## DayOfWeek                                       DayOfWeek 2.006794e-01
## Day                                                   Day 1.924131e-01
## HOLIDAY                                           HOLIDAY 1.425271e-01
## SUN                                                   SUN 7.972673e-02
## TUE                                                   TUE 8.861225e-03
## SAT                                                   SAT 8.073015e-03
## Quarter                                           Quarter 8.407026e-04
## Rank_Model1                                   Rank_Model1 0.000000e+00
## Rank_Model2                                   Rank_Model2 0.000000e+00
## Rank_Model3                                   Rank_Model3 0.000000e+00
## PAY                                                   PAY 0.000000e+00
## MON                                                   MON 0.000000e+00
## WED                                                   WED 0.000000e+00
## THU                                                   THU 0.000000e+00
```

```
## Before_holiday                                          Before_holiday 0.000000e+00
## After_holiday                                            After_holiday 0.000000e+00
```



Summary of Models

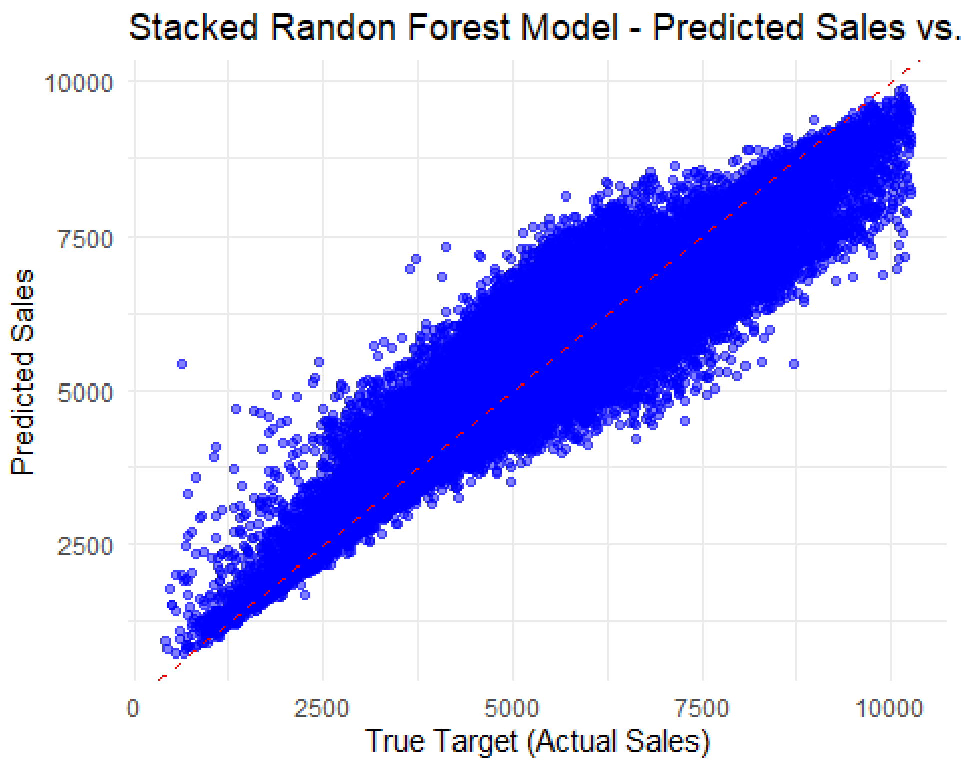
Evaluation_Metric	Stacked_Linear	Stacked_GBM
RMSE	1118.10873	1013.85987
MAPE	18.72952	16.29512
Forecast Bias	0.00000	-20.63712
Forecast Accuracy	85.07572	86.75844

The results of the stacked GBM model suggest that the forecasted values are approximately 1,000 units away from the actual values, the forecasted values deviate from the actual values by about 16%, the forecast is just slightly under-forecasting the actual values, and approximately 87% of the forecasted values are accurate compared to the actual values. Not only is this stacked model GBM model an improvement over the base models, it is also a slight improvement over the stacked LR model.

Step 5c. Generate predictions - Stacked Random Forest

Now that both the stacked LR and stacked GBM models are complete, we will attempt one more stacked model, this time using the Random Forest (RF) method.

```
##
## Call:
##  randomForest(formula = stacked_rf_formula, data = meta_input)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 9
##
##           Mean of squared residuals: 922607.3
##                     % Var explained: 68.46
```



Summary of Models

Evaluation_Metric	Stacked_Linear	Stacked_GBM	Stacked_RF
RMSE	1118.10873	1013.85987	553.253532
MAPE	18.72952	16.29512	7.356060
Forecast Bias	0.00000	-20.63712	3.119288
Forecast Accuracy	85.07572	86.75844	93.451912

The results of the stacked RF model display the lowest RMSE & MAPE and the highest accuracy of all stacked models, with minimal bias. The stacked RF model is performing the best of all stacked models. However, the evaluation metrics are favorable for all three stacked models, so we decide to test each stacked model against the test data.

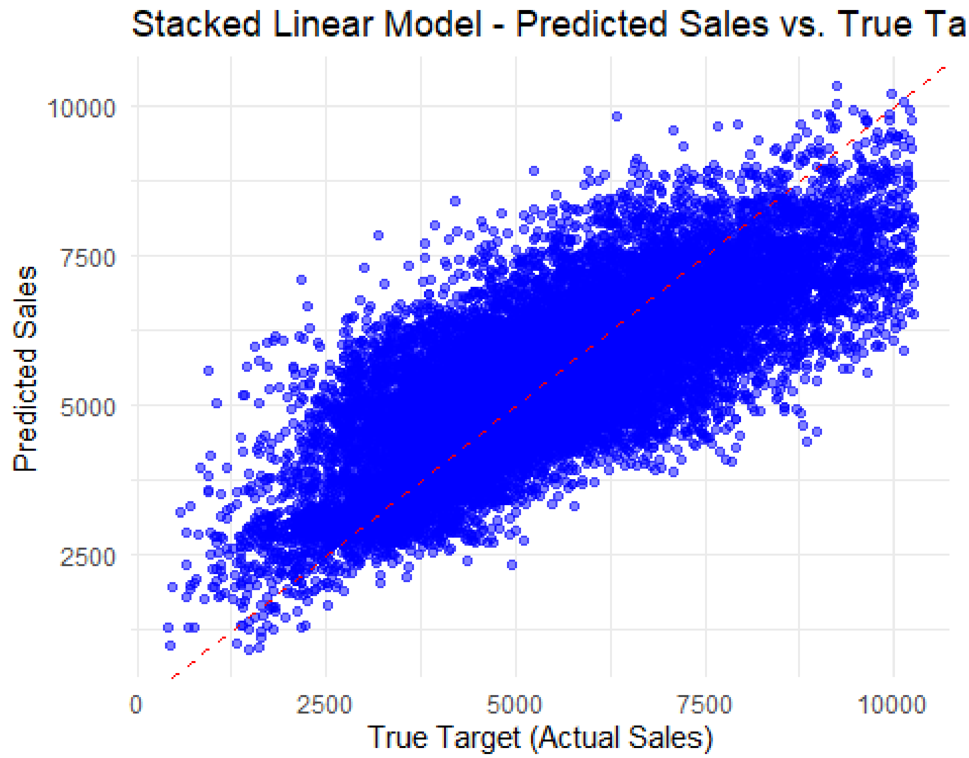
Step 6. Generate predictions on the test data

We will now use the trained meta-model to generate final ensemble predictions on the test data. We will employ a similar five step process to generate the predictions.
In order to reduce potential for errors, all of the vectors for the meta_input are retaining the same naming structure as the original meta_input.

Meta-model on test data: Linear Regression Model

```
##
## Call:
## lm(formula = linear_formula, data = meta_input)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4915.2  -725.4    -3.8   708.1  4446.6
##
## Coefficients: (3 not defined because of singularities)
##                                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)                     2.549e+06  2.534e+05  10.058  < 2e-16 ***
## Date                           -1.322e+02  1.311e+01 -10.080  < 2e-16 ***
## Prediction_Model1              -4.037e+00  4.389e-01  -9.197  < 2e-16 ***
## Prediction_Model2              -5.647e+00  4.746e-01 -11.898  < 2e-16 ***
## Prediction_Model3              -4.677e+00  4.463e-01 -10.478  < 2e-16 ***
## Rank_Model1                     4.096e-02  9.728e-03   4.210 2.56e-05 ***
## Rank_Model2                    -4.470e-02  2.006e-03 -22.288  < 2e-16 ***
## Rank_Model3                     1.018e-01  9.720e-03  10.478  < 2e-16 ***
## Difference_Model1_Model2       -1.012e+00  1.268e-01  -7.983 1.52e-15 ***
## Difference_Model1_Model3       -2.966e-01  1.139e-01  -2.605  0.00918 **
## Difference_Model2_Model3        9.995e-02  1.128e-01   0.886  0.37578
## Geometric_Mean_Predictions      1.529e+01  1.253e+00  12.209  < 2e-16 ***
## Standard_Deviation_Predictions  1.873e+00  3.380e-01   5.543 3.02e-08 ***
## Median_Predictions              7.397e-01  1.577e-01   4.690 2.75e-06 ***
## HOLIDAY                         2.413e+02  4.602e+01   5.242 1.61e-07 ***
## PAY                            -3.128e+02  5.270e+01  -5.935 2.98e-09 ***
## MON                            -1.079e+02  3.309e+01  -3.260  0.00112 **
## TUE                            -1.396e+02  3.585e+01  -3.894 9.88e-05 ***
## WED                            -3.285e+02  3.877e+01  -8.473  < 2e-16 ***
## THU                            -3.296e+02  4.564e+01  -7.221 5.34e-13 ***
## FRI                            -4.497e+02  6.149e+01  -7.314 2.71e-13 ***
## SAT                            -3.223e+02  4.295e+01  -7.503 6.54e-14 ***
## SUN                                    NA         NA      NA       NA
## Month                           4.032e+03  3.985e+02  10.116  < 2e-16 ***
## Day                             1.602e+02  1.309e+01  12.243  < 2e-16 ***
## Quarter                        -9.115e+01  3.287e+01  -2.773  0.00557 **
## DayOfWeek                              NA         NA      NA       NA
## Days_until_next_payday          2.994e+01  1.246e+00  24.032  < 2e-16 ***
## Before_holiday                         NA         NA      NA       NA
## After_holiday                   3.173e+01  2.669e+01   1.189  0.23450
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## Residual standard error: 1158 on 18229 degrees of freedom
## Multiple R-squared:  0.5978, Adjusted R-squared:  0.5973
## F-statistic:  1042 on 26 and 18229 DF,  p-value: < 2.2e-16
```



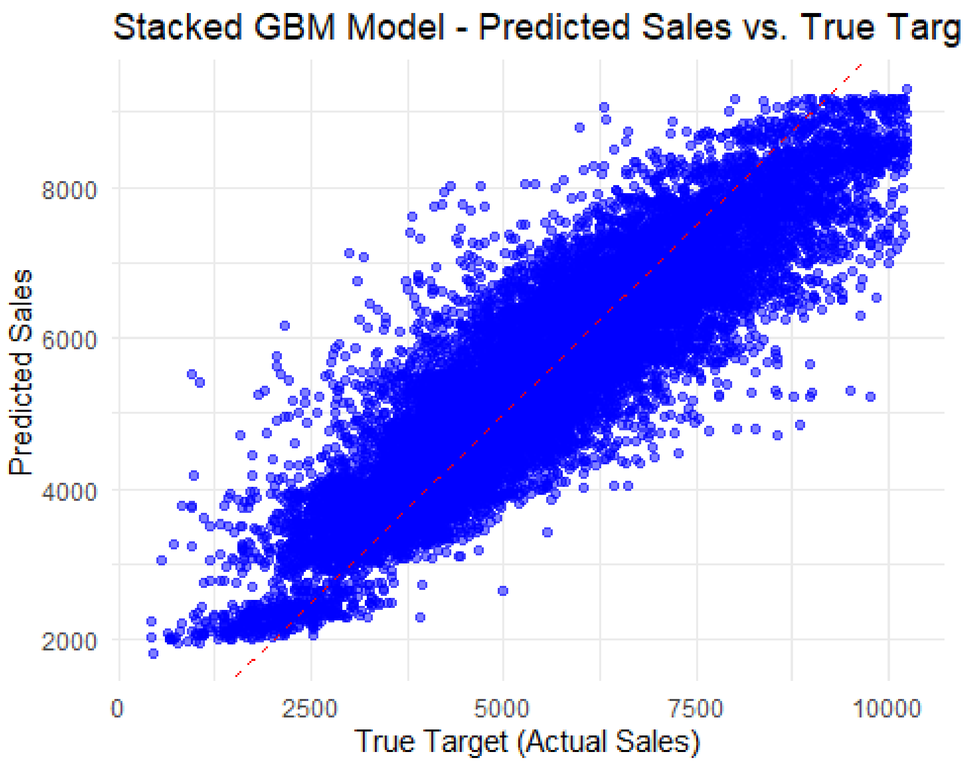
Summary of Models – Test Data

Evaluation_Metric	Stacked_Linear
RMSE	1157.32197
MAPE	18.91861
Forecast Bias	0.00000
Forecast Accuracy	84.94792

We see that the stacked Linear model evaluation metrics on testing data are nearly identical to the evaluation metrics of the Linear model on training data. This is a positive indication of the model's performance and generalization ability. This indicates that the model has learned the underlying patterns and relationships in the data rather than simply memorizing the training set.

Meta-model on test data: Gradient Boosting Machines Model

```
##                                                             var      rel.inf
## Prediction_Model3                             Prediction_Model3 52.151527483
## Prediction_Model2                             Prediction_Model2 18.610437060
## Geometric_Mean_Predictions           Geometric_Mean_Predictions 10.704260614
## Median_Predictions                           Median_Predictions  8.695368786
## Difference_Model2_Model3               Difference_Model2_Model3  5.141821185
## Days_until_next_payday                   Days_until_next_payday  1.796367698
## Month                                                     Month  0.873461601
## Prediction_Model1                             Prediction_Model1  0.626697389
## Difference_Model1_Model2               Difference_Model1_Model2  0.607474511
## Standard_Deviation_Predictions Standard_Deviation_Predictions  0.258645866
## Day                                                         Day  0.132711901
## DayOfWeek                                             DayOfWeek  0.130326381
## Difference_Model1_Model3               Difference_Model1_Model3  0.124501189
## WED                                                         WED  0.049693488
## SUN                                                         SUN  0.049688174
## Quarter                                                 Quarter  0.030534392
## SAT                                                         SAT  0.012883551
## After_holiday                                     After_holiday  0.003598731
## Rank_Model1                                         Rank_Model1  0.000000000
## Rank_Model2                                         Rank_Model2  0.000000000
## Rank_Model3                                         Rank_Model3  0.000000000
## HOLIDAY                                                 HOLIDAY  0.000000000
## PAY                                                         PAY  0.000000000
## MON                                                         MON  0.000000000
## TUE                                                         TUE  0.000000000
## THU                                                         THU  0.000000000
## FRI                                                         FRI  0.000000000
## Before_holiday                                   Before_holiday  0.000000000
```



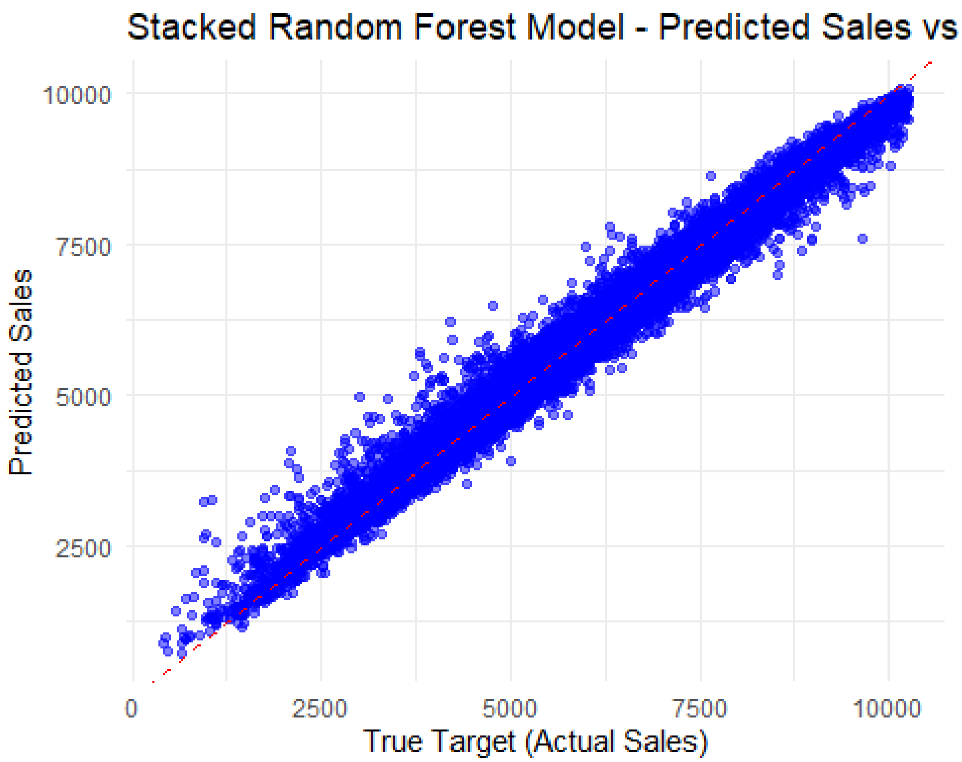
Summary of Models – Test Data

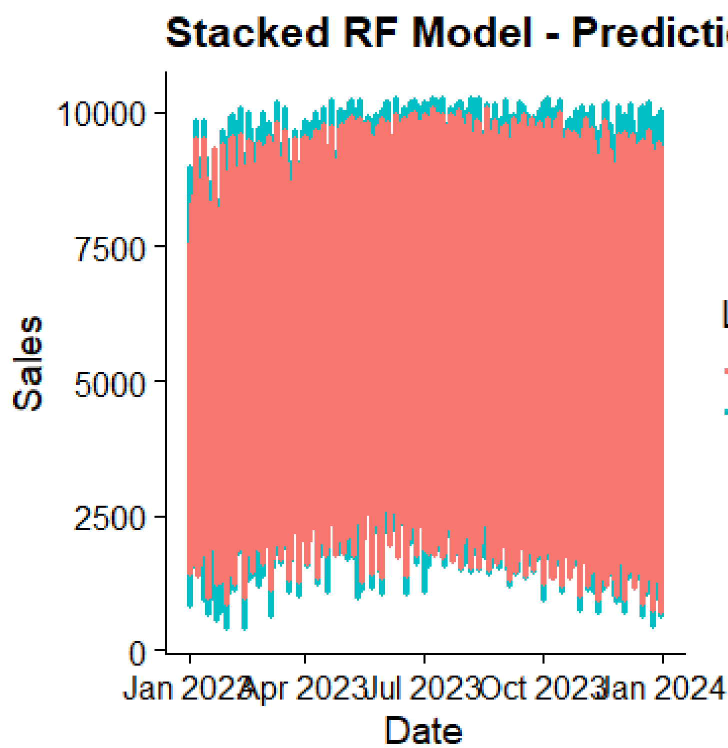
Evaluation_Metric	Stacked_Linear	Stacked_GBM
RMSE	1157.32197	809.563257
MAPE	18.91861	13.027538
Forecast Bias	0.00000	1.628157
Forecast Accuracy	84.94792	89.111282

The evaluation metrics of the stacked GBM model on the testing data are slightly improved compared to the evaluation metrics of the stacked GBM model on training data. This suggests that the model is generalizing well to unseen data and that adjustments made during testing have led to a slightly better performance.

Meta-model on test data: Random Forest Model

```
##
## Call:
##  randomForest(formula = stacked_rf_formula, data = meta_input)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 9
##
##           Mean of squared residuals: 366902.9
##                     % Var explained: 88.98
```


Stacked Random Forest Model - Predicted Sales vs
Predicted Sales
10000
7500
5000
2500
0
2500
5000
7500
10000
True Target (Actual Sales)


Stacked RF Model - Predictions vs True
Sales
10000
7500
5000
2500
0
Jan 2023
Apr 2023
Jul 2023
Oct 2023
Jan 2024
Date
Legend
Predictions
Target

Summary of Models – Test Data

Evaluation_Metric	Stacked_Linear	Stacked_GBM	Stacked_RF
RMSE	1157.32197	809.563257	270.234238
MAPE	18.91861	13.027538	4.063737
Forecast Bias	0.00000	1.628157	2.822545
Forecast Accuracy	84.94792	89.111282	96.258253

Similar to the stacked GBM model, the evaluation metrics of the stacked RF model on the testing data are slightly improved compared to the evaluation metrics of the stacked RF model on training data. This suggests that the model is generalizing well to unseen data and that adjustments made during testing have led to a slightly better performance.

Conclusions

In conclusion, our new sales forecasting models have demonstrated significant enhancements in sales forecasting compared to the current linear regression method employed by the company. Through the incorporation of additional relevant predictor variables and the exploration of various time series models, alongside newer machine learning techniques, we have identified models that outperform the original approach.

Specifically, the ARIMA, Prophet, and GBM models showed improvements over the naive baseline. While the new LR model with third-party data and the STLF model did not yield substantial enhancements, the Random Forest and GBM machine learning models reduced deviations with minimal bias, albeit with lower accuracy than traditional methods.

To enhance our predictions, we adopted an ensemble stacking approach, combining the strengths of the linear regression, ARIMA, and GBM models. The stacked ensemble methods demonstrated improved performance compared to the individual models, with the stacked Random Forest model emerging as the most promising option.

Upon evaluation with the unseen test data, all three stacked ensemble methods maintained strong performance, with the Random Forest model exhibiting the best evaluation metrics.

Based on these findings, we recommend the company adopt the Stacked Ensemble Random Forest model for sales forecasting. This model not only improves upon the limitations of the current linear regression method but also offers enhanced accuracy and effectiveness in predicting sales trends, thereby aiding the company in making more informed business decisions.

References

- https://mode.com/example-gallery/forecasting_prophet_r_cookbook
- https://rafalab.dfci.harvard.edu/dsbook/
- https://1965eric.github.io/Machine_Learning/

- https://www.kaggle.com/code/ekrembayar/store-sales-ts-forecasting-a-comprehensive-guide/notebook

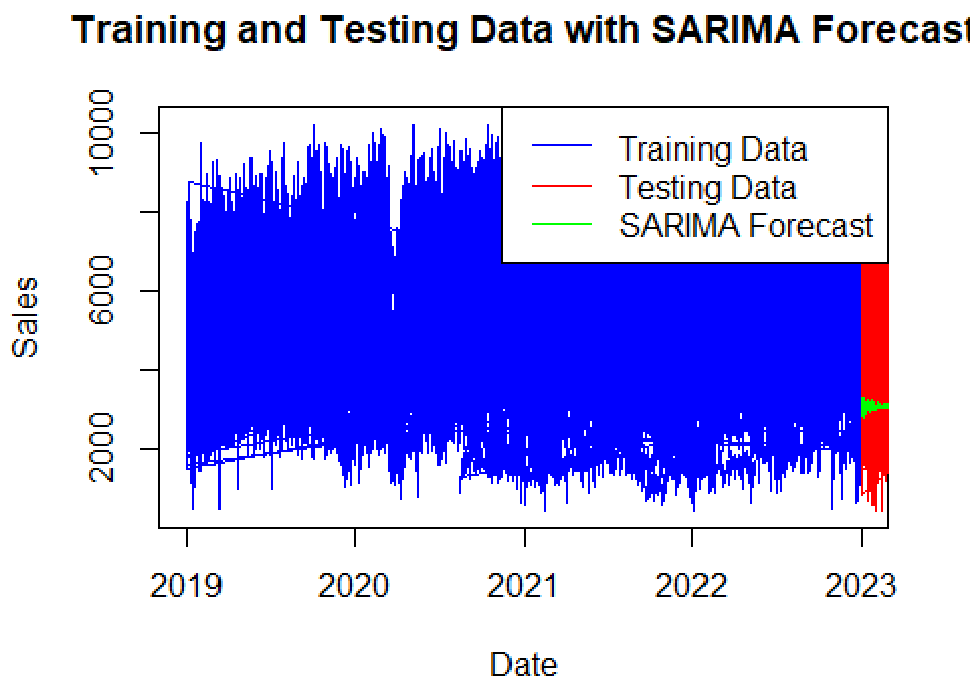
Appendix

Seasonal ARIMA Model (SARIMA)

The SARIMA model below delivered evaluation metrics with a high RMSE & MAPE, large forecast bias, and forecast accuracy that was no better than the Naive baseline forecast. Therefore, in order for this model to be effective, it would need to be tuned using the SARIMA-X method where X represents the external predictors. Additional packages like 'forecast' or 'fable' in conjunction with 'auto-arima()' are needed to build the SARIMA-X model where we specify both the time series data and the external predictors.

```
set.seed(123)

# Fit a SARIMA model
timing <- system.time({
  sarima_model <- auto.arima(sales_ts, seasonal=TRUE) #<- Warning - takes 8
to 10 min to run
  summary(sarima_model)
})
timing

##    user  system elapsed
##  477.36   10.32  490.67

length(testing_data$Sales)

## [1] 18256

# Generate forecasts using the SARIMA model for testing data
sarima_forecast_testing <- forecast(sarima_model, h = 18256)

# Extract forecasted sales values from the SARIMA forecast object
sarima_forecasted_values <- sarima_forecast_testing$mean

length(testing_data$Sales)

## [1] 18256

length(sarima_forecasted_values)

## [1] 18256

# Plot training data and testing data
plot(training_data$Date, training_data$Sales, type = "l", col = "blue", xlab
= "Date", ylab = "Sales", main = "Training and Testing Data with SARIMA
Forecast")
lines(testing_data$Date, testing_data$Sales, col = "red")
```

```r
# Plot SARIMA forecasted values
lines(testing_data$Date, sarima_forecasted_values, col = "green")

# Add legend
legend("topright", legend = c("Training Data", "Testing Data", "SARIMA
Forecast"), col = c("blue", "red", "green"), lty = 1)
```




```r
# Adjust plot limits
ylim <- range(training_data$Sales, testing_data$Sales,
sarima_forecasted_values)
ylim <- c(floor(ylim[1] / 1000) * 1000, ceiling(ylim[2] / 1000) * 1000)
ylim(ylim)

## <ScaleContinuousPosition>
##  Range:
##  Limits:    0 -- 1.1e+04

# Calculate evaluation metrics for the SARIMA forecast
rmse_sarima <- sqrt(mean((testing_data$Sales - sarima_forecasted_values)^2))
mape_sarima <- mean(abs((testing_data$Sales - sarima_forecasted_values) /
testing_data$Sales)) * 100
forecast_bias_sarima <- mean(testing_data$Sales - sarima_forecasted_values)
fa_sarima <- mean(1 - abs(testing_data$Sales - sarima_forecasted_values) /
pmax(testing_data$Sales, sarima_forecasted_values)) * 100

summary_table <- data.frame(
  Evaluation_Metric = c("RMSE", "MAPE", "Forecast Bias", "Forecast
```

```
Accuracy"),
  Naive = c(rmse_naive, mape_naive, forecast_bias, fa_naive),
  Linear = c(rmse_lm, mape_lm, forecast_bias_lm, fa_lm),
  ARIMA = c(rmse_arima, mape_arima, forecast_bias_arima, fa_arima),
  SARIMA = c(rmse_sarima, mape_sarima, forecast_bias_sarima, fa_sarima))

kable(summary_table, caption = "Summary of Models", align = "c")
```

Summary of Models

Evaluation_Metric	Naive	Linear	ARIMA	SARIMA
RMSE	3217.61010	1623.11461	1862.36789	3008.60065
MAPE	44.98366	31.32334	33.25723	41.24419
Forecast Bias	2650.00927	0.00000	212.04841	2391.93933
Forecast Accuracy	55.91194	78.32954	76.23366	59.98921