An Empirical Study of Time Series Forecasting Methods on Retail Store Sales Data

Ajith Kumar Ethirajulu, Deepak Raj Mohan Raj, Madhu Sikha Babu, Zeming Zhang

2023-04-26

Introduction:

The retail industry heavily relies on accurate sales forecasting to optimize inventory management, plan promotions, and make informed business decisions. In this project, we conducted an empirical study to forecast retail store sales using various time series forecasting methods. Our dataset comprises a vast collection of 3,000,888 records, capturing sales details from 54 distinct stores and 83,488 transactions.

The primary objective of this study was to analyze the components of sales, such as trend and seasonality, and evaluate the performance of different forecasting models implemented in R programming. We employed seasonal naive, exponential smoothing (ETS), ARIMA, and Prophet models, assessing their ability to capture underlying patterns in the sales data and provide accurate predictions. Key evaluation metrics, including the root mean square error (RMSE), were used to compare the forecasting accuracy of these models.

```
knitr::opts_chunk$set(echo = TRUE, warning = FALSE)

# installing the necessary libraries
library(vctrs)
library(zoo)
library(tibble)
library(dplyr)
library(tidyr)
library(readr)
library(lubridate)
library(ggplot2)
library(tsibble)
library(fable)
library(fabletools)
library(feasts)
library(tsibbledata)
library(cowplot)
library(rlang)
library(prophet)
library(plotly)
library(RColorBrewer)
```

```
# Read the data as dataframe
train <- read.csv("train.csv")
test <- read.csv("test.csv")
stores <- read.csv("stores.csv")
head(train)
```

	id	date	store_nbr	family	sales	onpromotion
	<int>	<chr>	<int>	<chr>	<dbl>	<int>
1	0	2013-01-01	1	AUTOMOTIVE	0	0
2	1	2013-01-01	1	BABY CARE	0	0
3	2	2013-01-01	1	BEAUTY	0	0
4	3	2013-01-01	1	BEVERAGES	0	0
5	4	2013-01-01	1	BOOKS	0	0
6	5	2013-01-01	1	BREAD/BAKERY	0	0

6 rows

```
transactions <- read.csv("transactions.csv") %>% arrange(store_nbr, date)

# Datetime Conversion
train$date <- as.Date(train$date, format = "%Y-%m-%d")
test$date <- as.Date(test$date, format = "%Y-%m-%d")
transactions$date <- as.Date(transactions$date, format = "%Y-%m-%d")

# Data types conversion
train$onpromotion <- as.numeric(train$onpromotion)
train$sales <- as.numeric(train$sales)
stores$cluster <- as.integer(stores$cluster)

head(train)
```

	id	date	store_nbr	family	sales	onpromotion
	<int>	<date>	<int>	<chr>	<dbl>	<dbl>
1	0	2013-01-01	1	AUTOMOTIVE	0	0
2	1	2013-01-01	1	BABY CARE	0	0
3	2	2013-01-01	1	BEAUTY	0	0
4	3	2013-01-01	1	BEVERAGES	0	0
5	4	2013-01-01	1	BOOKS	0	0
6	5	2013-01-01	1	BREAD/BAKERY	0	0

6 rows

```
head(transactions)
```

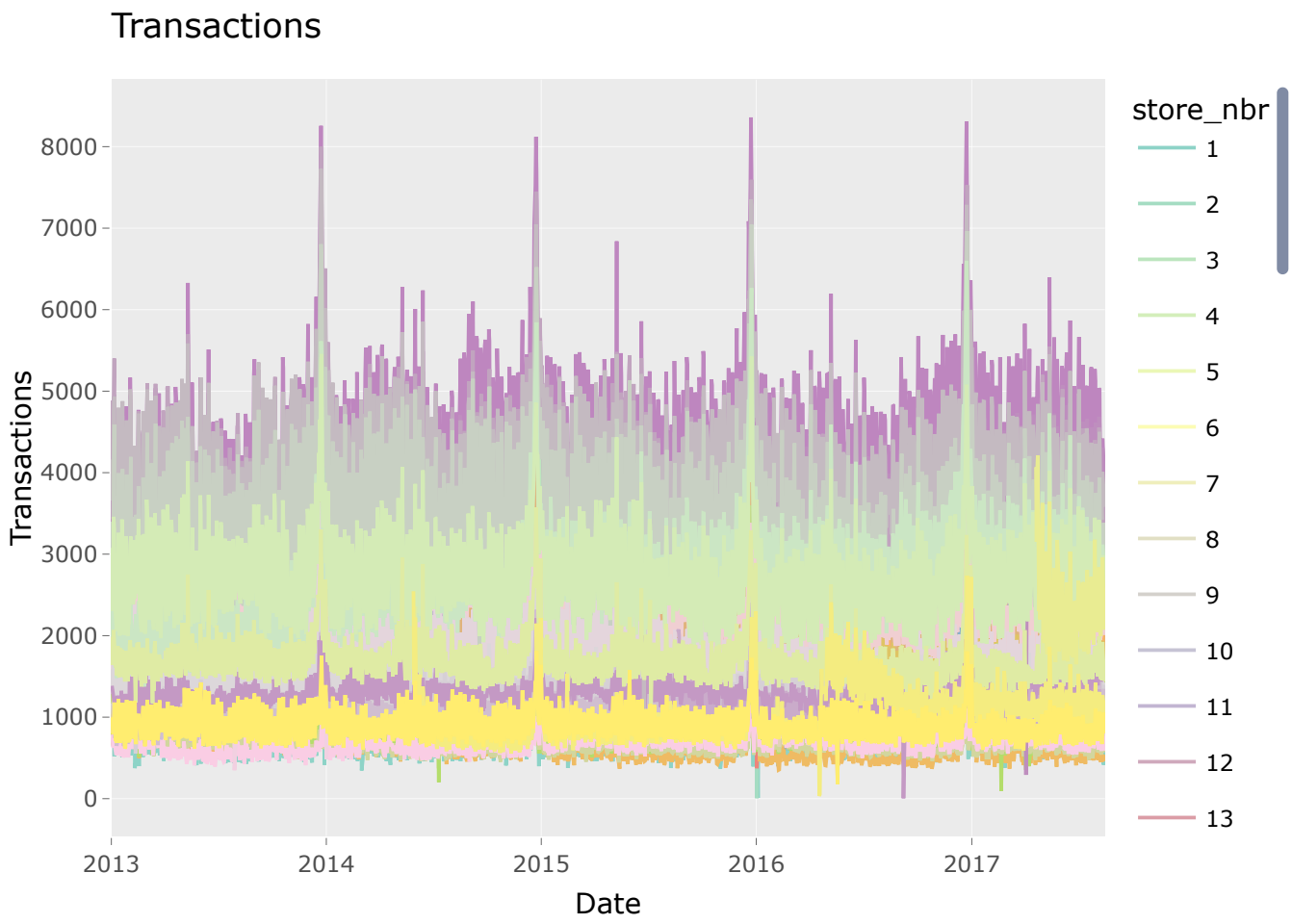
	date	store_nbr	transactions
	<date>	<int>	<int>
1	2013-01-02	1	2111
2	2013-01-03	1	1833
3	2013-01-04	1	1863
4	2013-01-05	1	1509
5	2013-01-06	1	520
6	2013-01-07	1	1807

6 rows

```
# Generate a color palette with 54 colors
my_colors <- colorRampPalette(brewer.pal(12, "Set3"))(54)

# Merge data frames
temp <- train %>%
  group_by(date, store_nbr) %>%
  summarize(sales = sum(sales), .groups = "drop") %>%
  left_join(transactions, by = c("date", "store_nbr"))

# Calculate Spearman correlation and print the result
corr <- cor(temp$sales, temp$transactions, method = "spearman", use = "complete.obs")
cat("Spearman Correlation between Total Sales and Transactions: ", formatC(corr, format
= "f", digits = 4), "\n")
```

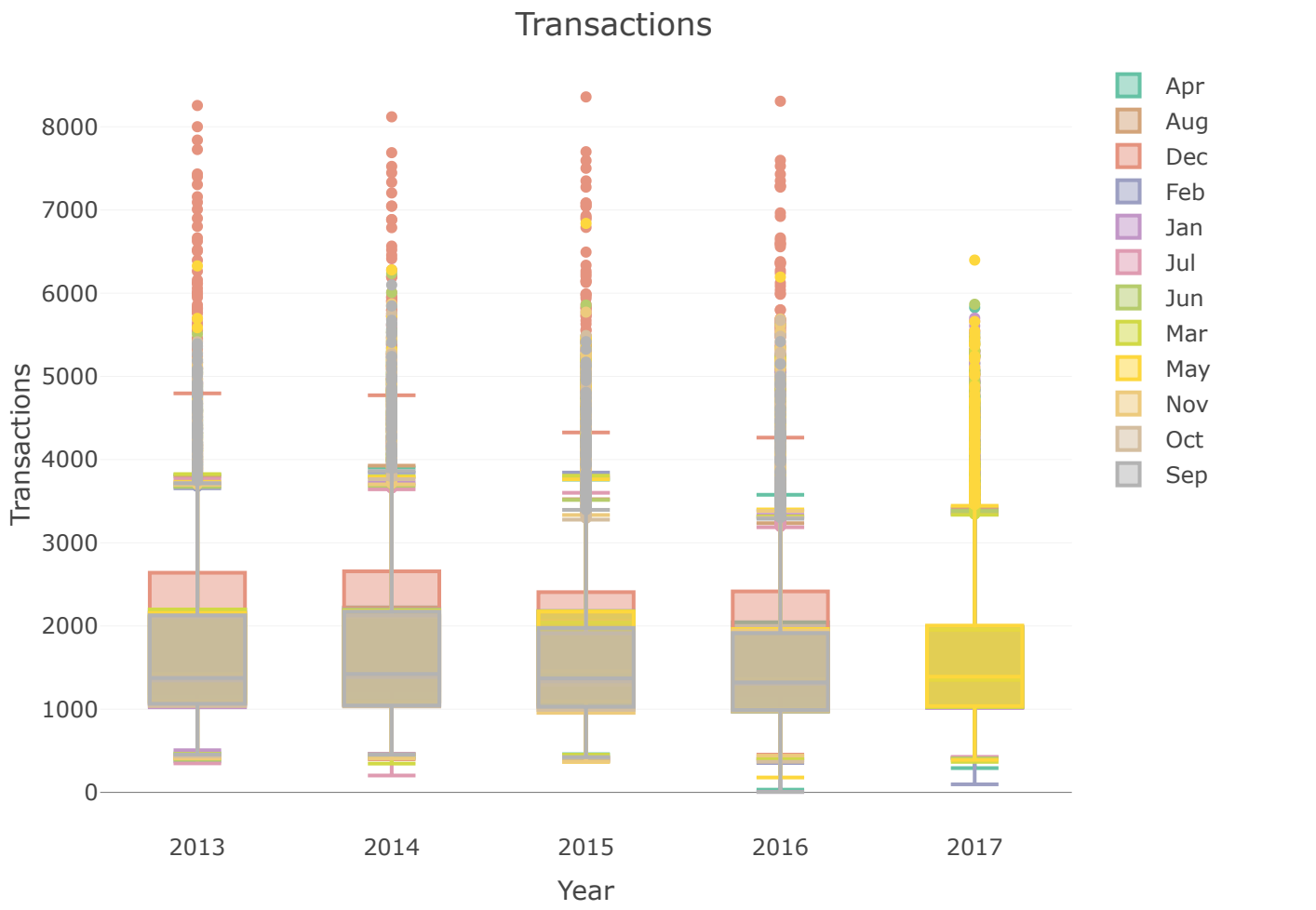
```
## Spearman Correlation between Total Sales and Transactions:  0.8175
```

```
# Plot transactions by store
plot_data <- transactions %>%
  arrange(store_nbr, date) %>%
  mutate(store_nbr = as.factor(store_nbr)) %>%   # Convert to factor
  ggplot(aes(x = date, y = transactions, color = store_nbr)) +
  geom_line() +
  ggtitle("Transactions") +
  xlab("Date") +
  ylab("Transactions") +
  scale_color_manual(values = my_colors)

# Increase the width of the figure
options(repr.plot.width=10)

# Convert ggplot to plotly object
plot_data <- ggplotly(plot_data, tooltip = c("store_nbr", "transactions", "date"), dynam
icTicks = TRUE)

# Show the plot
plot_data
```



This is an interactive plot using ggplotly, which helps us to check the transactions done on a particular date and store number by hovering over it.

```
# Create a copy of transactions as 'a'
a <- transactions

# Extract year and month from the date column
a$year <- format(a$date, "%Y")
a$month <- format(a$date, "%b")

# Plot a box plot
plot_data <- plot_ly(a, x = ~year, y = ~transactions, color = ~month, type = "box") %>%
  layout(title = "Transactions", xaxis = list(title = "Year"), yaxis = list(title = "Tra
nsactions"))

# Show the plot
plot_data
```



This is an interactive box plot which shows the outliers for transactions.

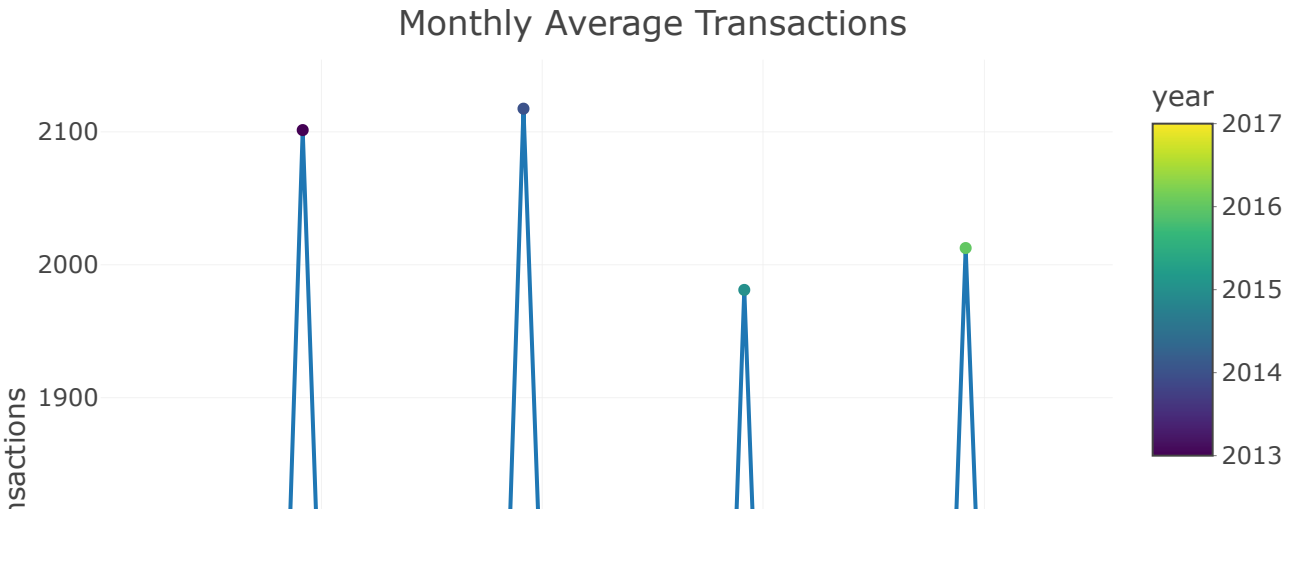
```
# Create a copy of transactions and resample to monthly frequency
a <- transactions %>%
  mutate(date = as.Date(date)) %>%
  group_by(year = year(date), month = month(date)) %>%
  summarize(transactions = mean(transactions), .groups = "drop") %>%
  mutate(date = ymd(paste(year, month, "01", sep = "-")))
head(a)
```

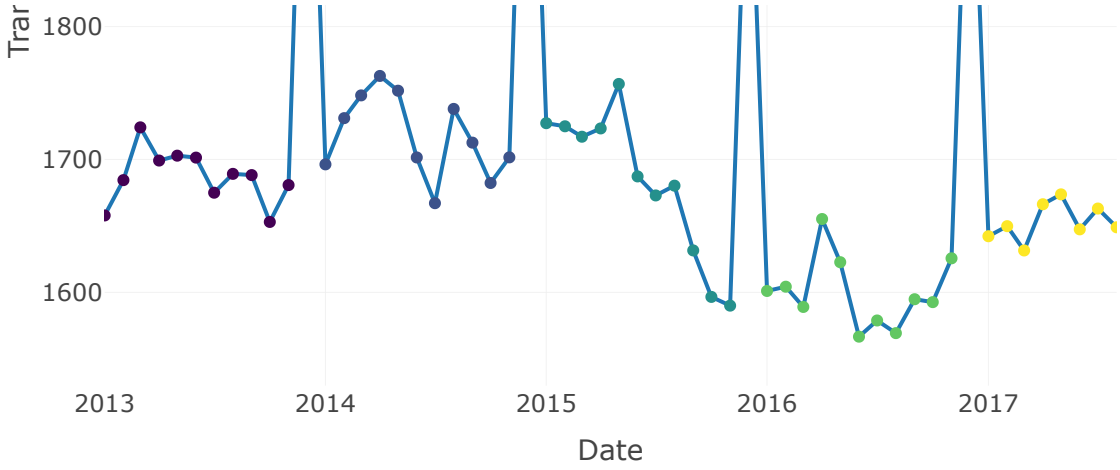
year <dbl>	month <dbl>	transactions <dbl>	date <date>
2013	1	1657.899	2013-01-01
2013	2	1684.484	2013-02-01
2013	3	1724.182	2013-03-01
2013	4	1699.207	2013-04-01
2013	5	1702.874	2013-05-01
2013	6	1701.427	2013-06-01

6 rows

```
# Define a color palette with a different color for each year
color_palette <- c("#1f77b4", "#ff7f0e", "#2ca02c", "#d62728", "#9467bd", "#8c564b", "#e
377c2", "#7f7f7f", "#bcbd22", "#17becf")

# Set the x-axis range to the minimum and maximum dates in the 'a' data frame
x_range <- range(a$date)

# Plot the data using plot_ly function
plot_data <- plot_ly(a, x = ~date, y = ~transactions, color = ~year, type = "scatter", m
ode = "lines",
                     marker = list(colors = color_palette))

# Add chart title and axis labels
plot_data <- plot_data %>% layout(title = "Monthly Average Transactions",
                                  xaxis = list(title = "Date", range = x_range),
                                  yaxis = list(title = "Transactions"))

# Show the plot
plot_data
```

```
## A marker object has been specified, but markers is not in the mode
## Adding markers to the mode...
```





This interactive plot shows the mean transactions each month, helps us to identify seasonal pattern.

```
# SELECTING A PARTICULAR STORE AND CATEGORY (FAMILY)

subset_train <- function(store, family_name) {
  train %>%
    filter(store_nbr == store & family == family_name) %>%
    select(-id, -store_nbr, -family, -onpromotion) %>%
    slice(1:nrow(.)) %>%
    as.data.frame()
}

df_sel <- subset_train(1, "GROCERY I")

head(df_sel)
```

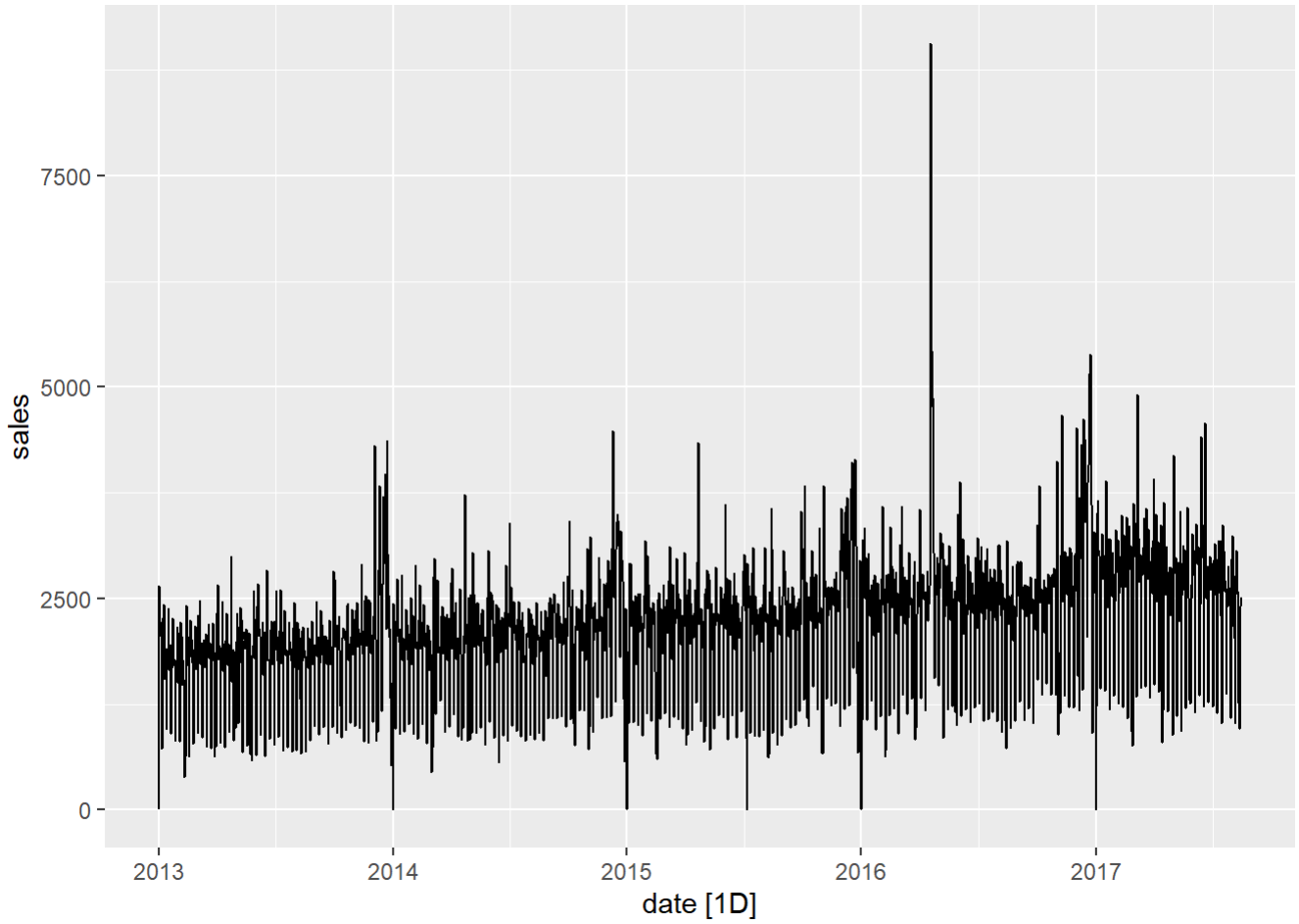
	date <date>	sales <dbl>
1	2013-01-01	0
2	2013-01-02	2652
3	2013-01-03	2121
4	2013-01-04	2056
5	2013-01-05	2216
6	2013-01-06	723

6 rows

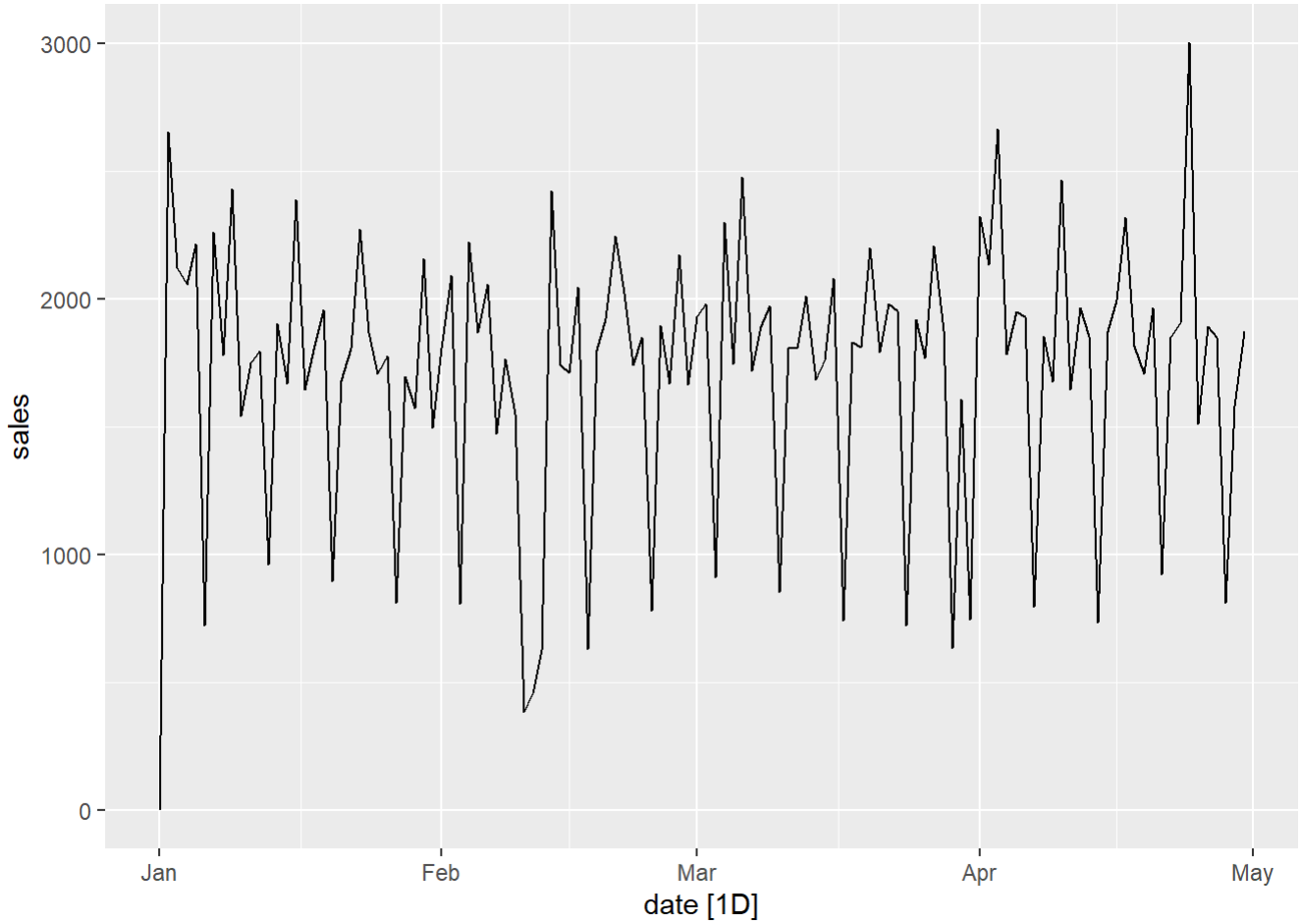
We have filtered the dataset for a particular store and Family (category) to analyse its sales pattern.

```
# convert to tsibble
df_sel %>% mutate(date=as_date(date)) %>% tsibble(index=date) ->
  df_sel
autoplot(df_sel)
```

```
## Plot variable not specified, automatically selected `.vars = sales`
```




```
df_sel_display <- df_sel %>% filter(date < as_date("2013-05-01"))   # weekly pattern

autoplot(df_sel_display, sales)
```



This plot shows the weekly seasonality of sales data for the particular store and family.

```
# train test split
df_sel_train <- df_sel %>% fill_gaps(sales = 0) %>% filter(date < as_date("2016-08-15"))
# shops close on christmas, hence dates missing on dec 25th every year.
head(df_sel_train)
```

date <date>	sales <dbl>
2013-01-01	0
2013-01-02	2652
2013-01-03	2121
2013-01-04	2056
2013-01-05	2216
2013-01-06	723

6 rows

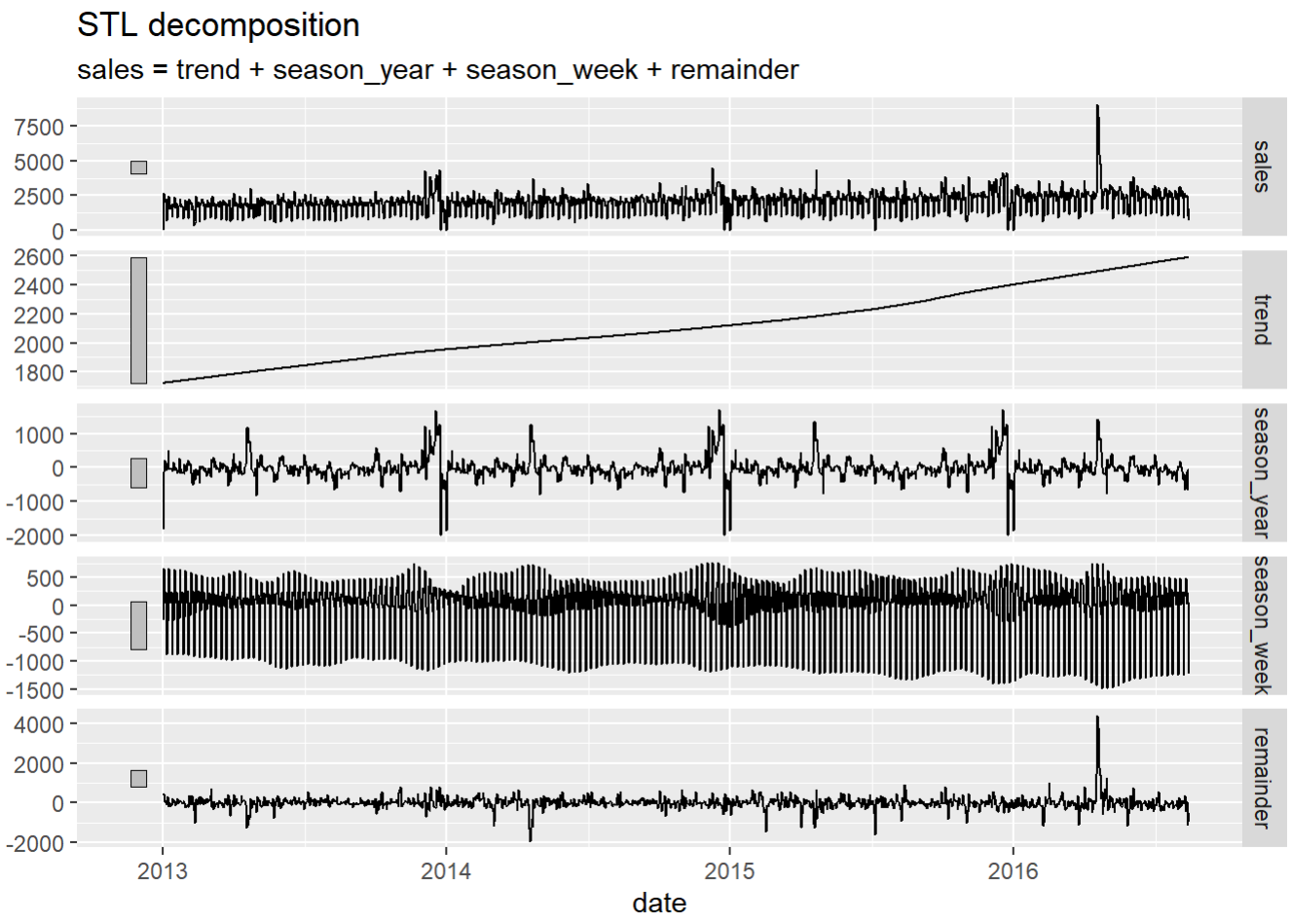
```
df_sel_test <- df_sel %>% fill_gaps(sales = 0) %>% filter(date >= as_date("2016-08-15"))
# replacing missing dates with sales 0
head(df_sel_test)
```

date	sales
<date>	<dbl>
2016-08-15	3181
2016-08-16	2436
2016-08-17	2422
2016-08-18	2201
2016-08-19	2538
2016-08-20	2308

6 rows

STL Decomposition

```
# STL decomposition to view the trend and seasonality components
df_sel_train %>% model(STL(sales)) %>% components() %>% autoplot()
```



This plot helps us visualize the different components of the patterns like trend and multiple seasonal components.

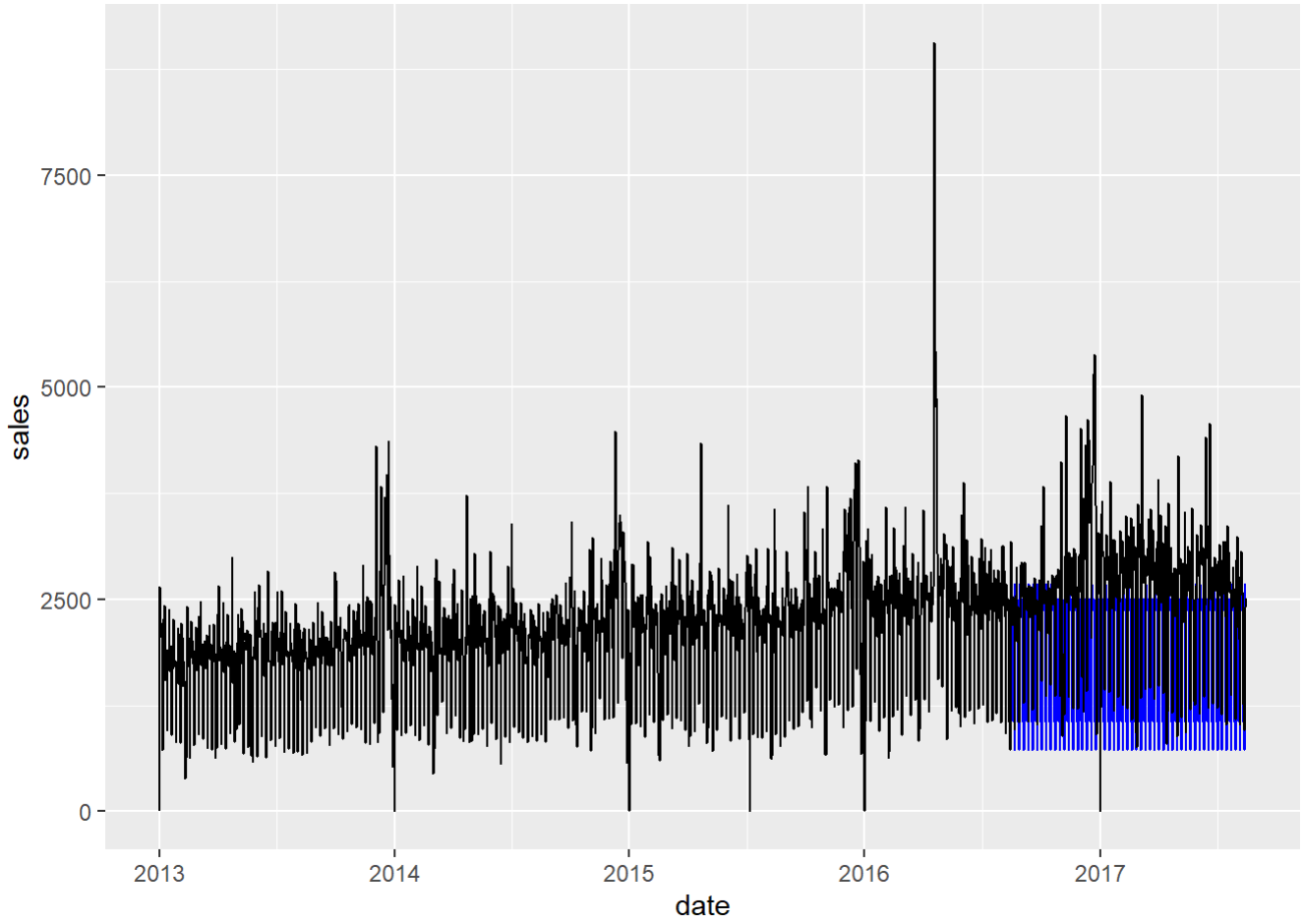
Fitting with Mean, Naive, Seasonal Naive and Drift models

```
# fitting basic models like mean, naive, seasonal naive and drift models for baseline co
mparison
fit <- df_sel_train %>%
  model(
    Mean = MEAN(sales),
    Naive = NAIVE(sales),
    Seasonal_Naive = SNAIVE(sales),
    Drift = RW(sales ~ drift())
  )
accuracy(fit)
```

.model	.type	ME	RMSE	MAE	M…	M…	MASE	RMSSE	
<chr>	<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	
Mean	Training	-1.928321e-13	740.7704	520.4077	-Inf	Inf	1.315037	1.064534	0
Naive	Training	5.510977e-01	895.4660	676.6495	-Inf	Inf	1.709850	1.286842	-0
Seasonal_Naive	Training	1.086692e+00	695.8633	395.7361	-Inf	Inf	1.000000	1.000000	0
Drift	Training	6.605020e-14	895.4658	676.6399	-Inf	Inf	1.709826	1.286841	-0

4 rows

```
#forecasting the sales using seasonal naive
fit <- fit %>% select(Seasonal_Naive)
fc <- fit %>% forecast(h = "1 year")
fc %>% autoplot(df_sel,level = NULL)
```

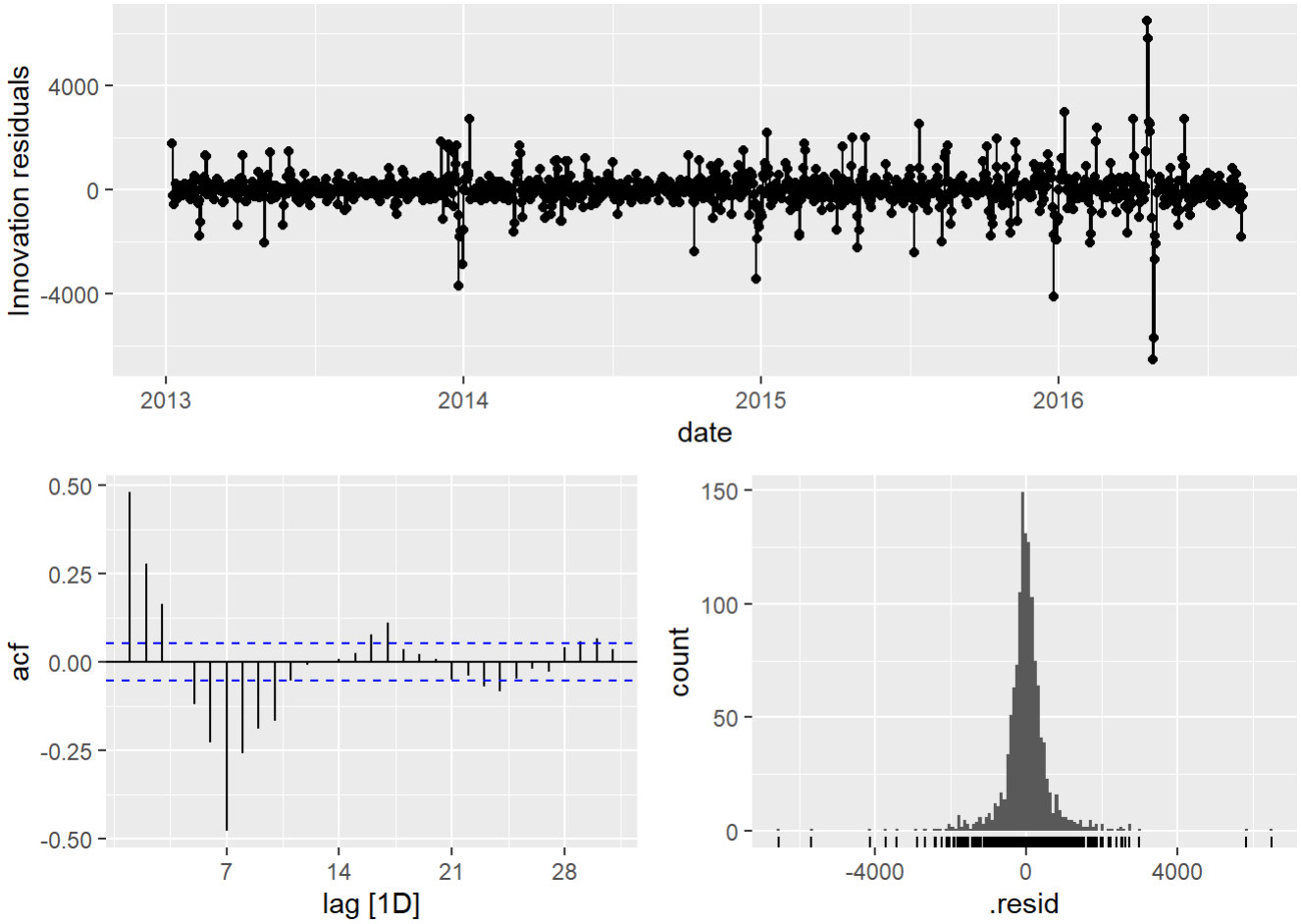



```
# printing test accuracy for seasonal naive model
accuracy(fc,df_sel)
```

.model	.type	ME	R…	MAE	M…	M…	MASE	RMSSE	ACF1
<chr>	<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
Seasonal_Naive	Test	683.3242	1068	799.7912	-Inf	Inf	2.013393	1.52938	0.1795886

1 row

```
# residual test for seasonal naive model forecast
gg_tsresiduals(fit)
```



Performing ETS model:

```
fit <- df_sel_train %>%
  model(
    ets_auto = ETS(sales),
    ets = ETS(sales ~ error("A") + trend("N") + season("A"))
  )
accuracy(fit)
```

.model	.type	ME	RMSE	MAE	M...	M...	MASE	RMSSE	ACF1
<chr>	<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
ets_auto	Training	-0.02773424	476.9298	280.3823	-Inf	Inf	0.7085082	0.6853786	0.1136142
ets	Training	-0.02773424	476.9298	280.3823	-Inf	Inf	0.7085082	0.6853786	0.1136142

2 rows

```
report(fit[1])
```

```
## Series: sales
## Model: ETS(A,N,A)
##   Smoothing parameters:
##     alpha = 0.417288
##     gamma = 0.01456816
##
##   Initial states:
##      l[0]     s[0]     s[-1]    s[-2]    s[-3]    s[-4]    s[-5]    s[-6]
##  1780.088 194.7795 -1026.544 111.9922 80.85093 2.186493 547.5536 89.18141
##
##   sigma^2:  229021.2
##
##      AIC     AICc      BIC
## 25827.61 25827.78 25879.48
```

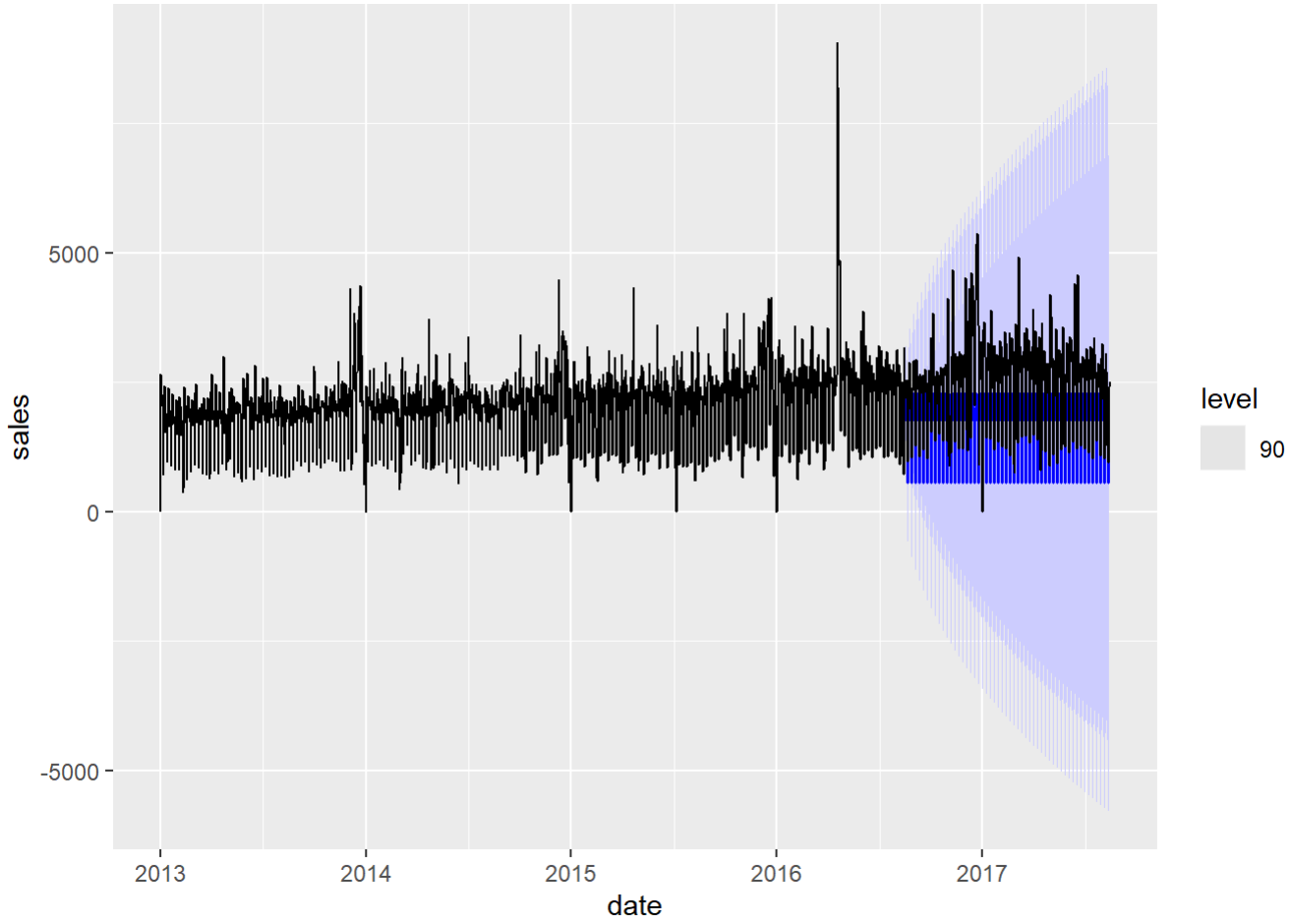
```
report(fit[2])
```

```
## Series: sales
## Model: ETS(A,N,A)
##   Smoothing parameters:
##     alpha = 0.417288
##     gamma = 0.01456816
##
##   Initial states:
##      l[0]     s[0]     s[-1]    s[-2]    s[-3]    s[-4]    s[-5]    s[-6]
##  1780.088 194.7795 -1026.544 111.9922 80.85093 2.186493 547.5536 89.18141
##
##   sigma^2:  229021.2
##
##      AIC     AICc      BIC
## 25827.61 25827.78 25879.48
```

```
fit <- fit %>% select(ets)
fc <- fit %>% forecast(h = "1 year")
```

Our chosen ETS model and the ETS auto model have turned out to be the same.

```
fc %>% autoplot(df_sel,level = 90)
```




```
accuracy(fc, df_sel)
```

.model	.type	ME	RMSE	MAE	MPE	M...	MASE	RMSSE	ACF1
<chr>	<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
ets	Test	842.6694	1020.906	903.9492	-Inf	Inf	2.2756	1.461941	0.3742548

1 row

From this accuracy, we can see that RMSE for ETS model is lesser than the seasonal naive model. This indicates better performance than seasonal naive.

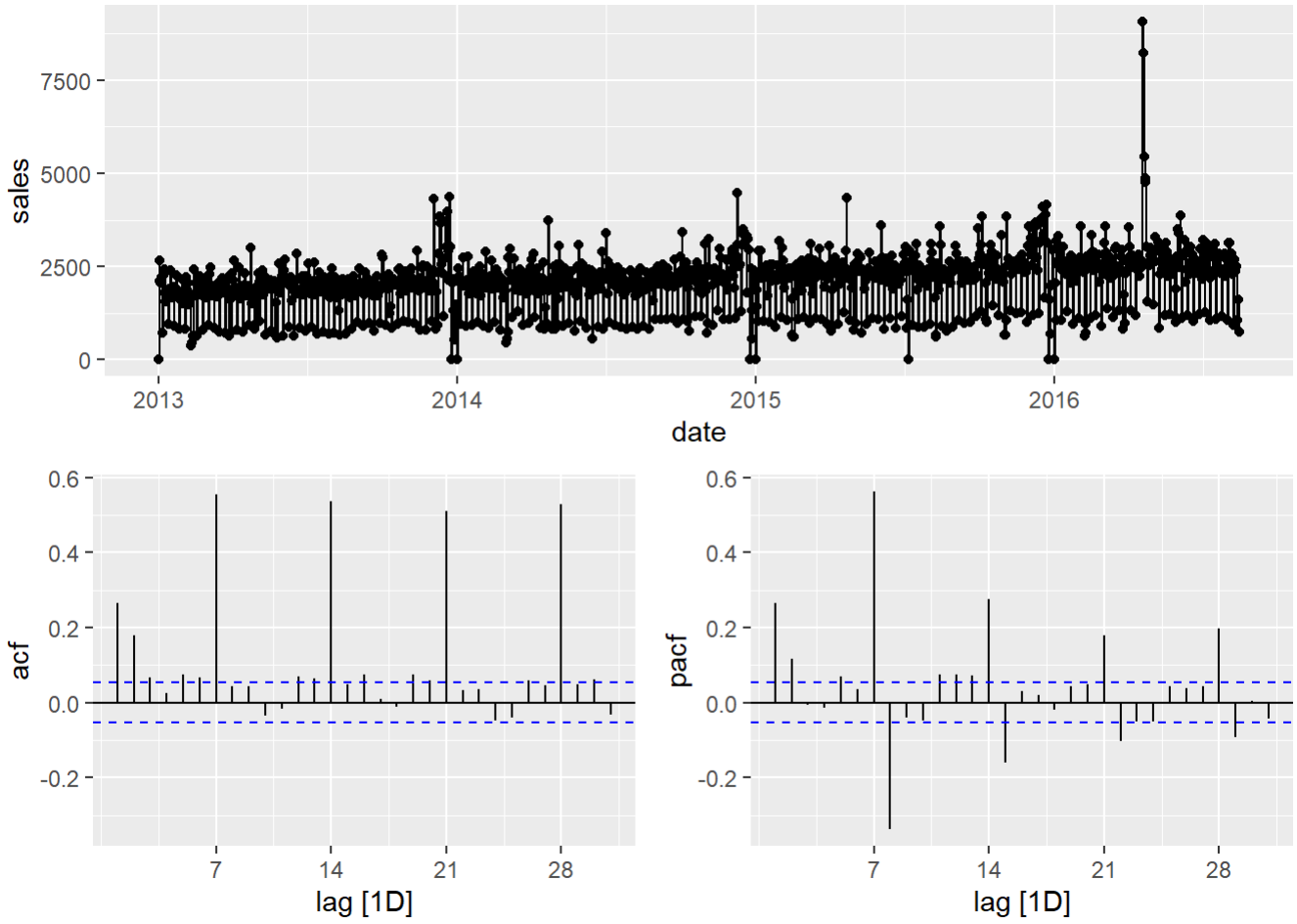
ARIMA

```
# performing kpss test
df_sel_train %>%features(sales, unitroot_kpss)
```

kpss_stat	kpss_pvalue
<dbl>	<dbl>
6.345942	0.01

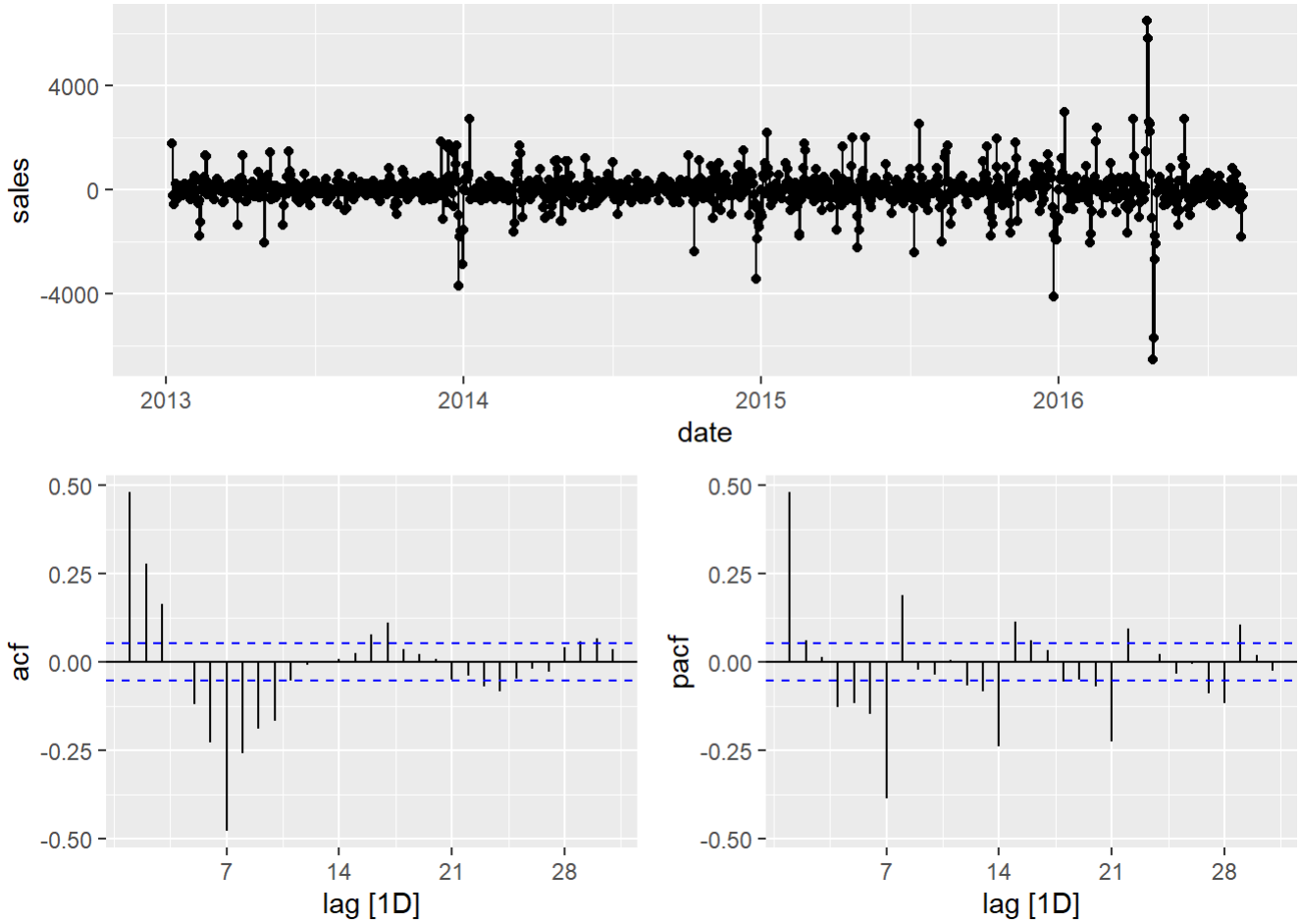
1 row

```
gg_tsdisplay(df_sel_train, sales, plot_type='partial')
```



ACF plot is not decreasing to zero. kpss pvalue is less 0.05. Hence null hypothesis is rejected. This indicates that data is not stationary, we need to perform differencing on the data.

```
# performing differencing
df_sel_train_diff <- df_sel_train %>% mutate(sales = difference(sales, lag = 7))

gg_tsdisplay(df_sel_train_diff, sales, plot_type='partial')
```




```
df_sel_train_diff <- drop_na(df_sel_train_diff, sales)

df_sel_train_diff %>%features(sales, unitroot_kpss)
```

kpss_stat	kpss_pvalue
<dbl>	<dbl>
0.007499888	0.1

1 row

After performing differencing, the ACF plot drops to zero sinusoidally/ exponentially. Also, the kpss pvalue is greater than 0.05. This indicates it is now converted to stationary data.

```
df_sel_train %>%features(difference(sales,lag=7), unitroot_ndiffs)
```

ndiffs
<int>
0

1 row

```
df_sel_train %>%features(difference(sales,lag=7), unitroot_nsdiffs)
```

nsdiffs
<int>
0

1 row

For differencing with lag = 7, we are getting nsdiffs and ndiffs as 0. This indicates we need to include D = 1 in ARIMA models.

From the ACF and PACF plot, based on our interpretations, we have to perform only seasonal differencing of period 1 (D=1), and the below setup for AR and MA.

(SEASONAL) ARIMA (p=1,d=0,q=3)(P=3,D=1,Q=1)

```
fit <- df_sel_train %>%
  model(
    arima_auto = ARIMA(sales, stepwise = FALSE, approx = FALSE),
    arima1 = ARIMA(sales~0+pdq(1,0,3)+PDQ(3,1,1))
  )
report(fit[1])
```

```
## Series: sales
## Model: ARIMA(0,0,4)(2,1,0)[7]
##
## Coefficients:
##          ma1     ma2     ma3     ma4     sar1     sar2
##       0.4433  0.2182  0.1880  0.1044  -0.5987  -0.2929
## s.e.  0.0278  0.0298  0.0285  0.0278   0.0267   0.0266
##
## sigma^2 estimated as 265775:  log likelihood=-10076.96
## AIC=20167.91   AICc=20168   BIC=20204.18
```

```
report(fit[2])
```

```
## Series: sales
## Model: ARIMA(1,0,3)(3,1,1)[7]
##
## Coefficients:
##          ar1      ma1      ma2     ma3    sar1     sar2     sar3     sma1
##       0.5495  -0.0950  -0.0026  0.0788  0.0459  -0.0197  -0.0549  -0.9554
## s.e.  0.1005   0.1013   0.0531  0.0375  0.0292   0.0288   0.0287   0.0089
##
## sigma^2 estimated as 203853:  log likelihood=-9908.81
## AIC=19835.62   AICc=19835.76   BIC=19882.25
```

```
glance(fit)
```

.model	sigma2	log_lik	AIC	AICc	BIC	ar_roots	ma_roots
<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<list>	<list>
arima_auto	265775.3	-10076.955	20167.91	20168.00	20204.18	<cpl [14]>	<cpl [4]>
arima1	203853.5	-9908.809	19835.62	19835.76	19882.25	<cpl [22]>	<cpl [10]>

2 rows

```
accuracy(fit)
```

.model	.type	ME	RMSE	MAE	M…	M…	MASE	RMSSE	AC
<chr>	<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<d
arima_auto	Training	2.330216	512.9929	316.1802	-Inf	Inf	0.7989673	0.7372036	-0.004807(
arima1	Training	40.976875	448.9326	264.0906	-Inf	Inf	0.6673402	0.6451448	-0.011068

2 rows

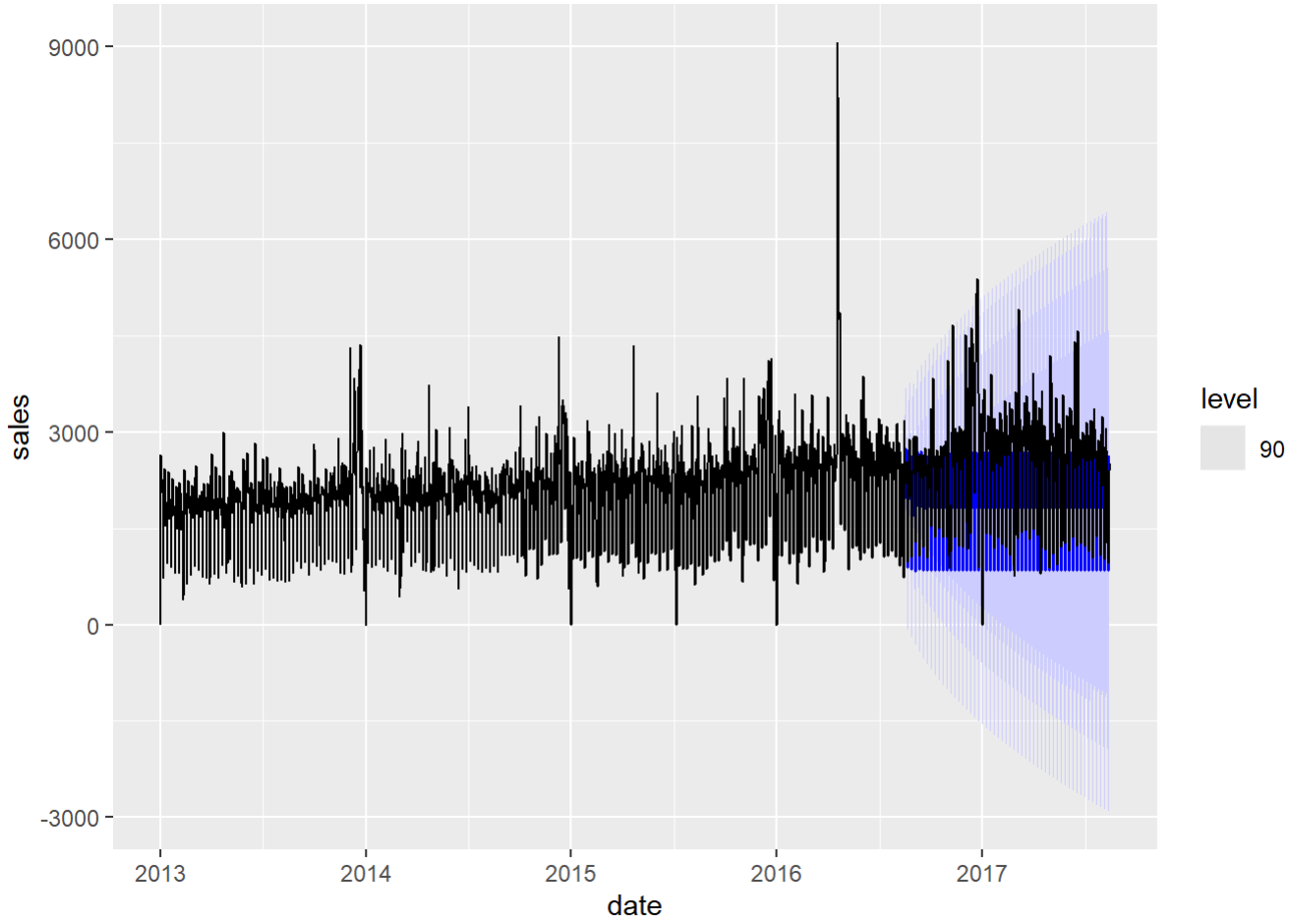
```
summary(forecast(fit$arima1, h = "1 year"))
```

```
##      Length Class  Mode
## [1,] 3      fbl_ts list
```

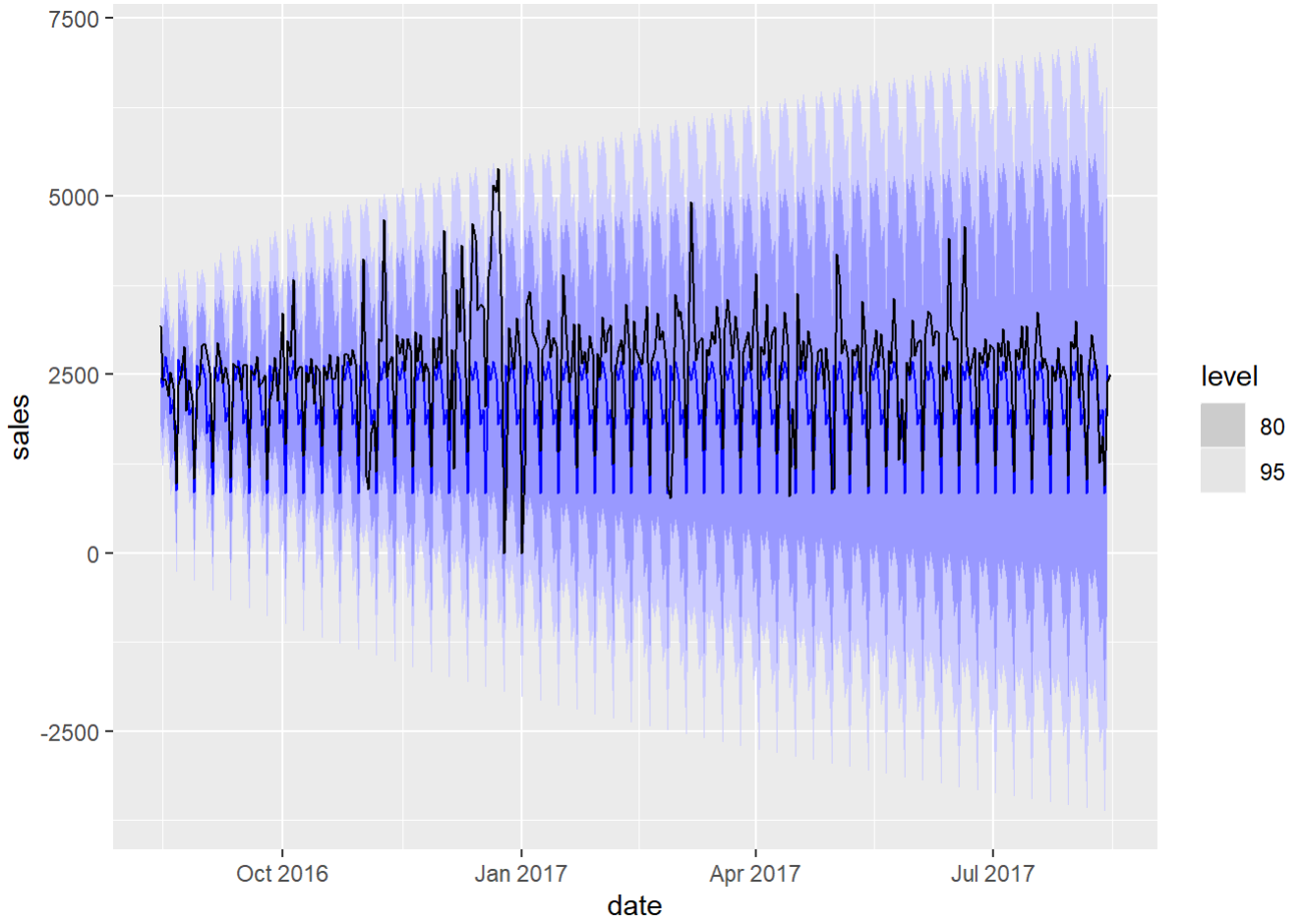
Based on the above results and summary, we can see the model that we chose is giving lesser AICc values and lesser RMSE values compared to the ARIMA auto model.

There is also a drastic decrease in the RMSE values in ARIMA model compared to ETS model and Seasonal Naive model. This concludes better performance of ARIMA (1 0 3)(3 1 1)[7] model.

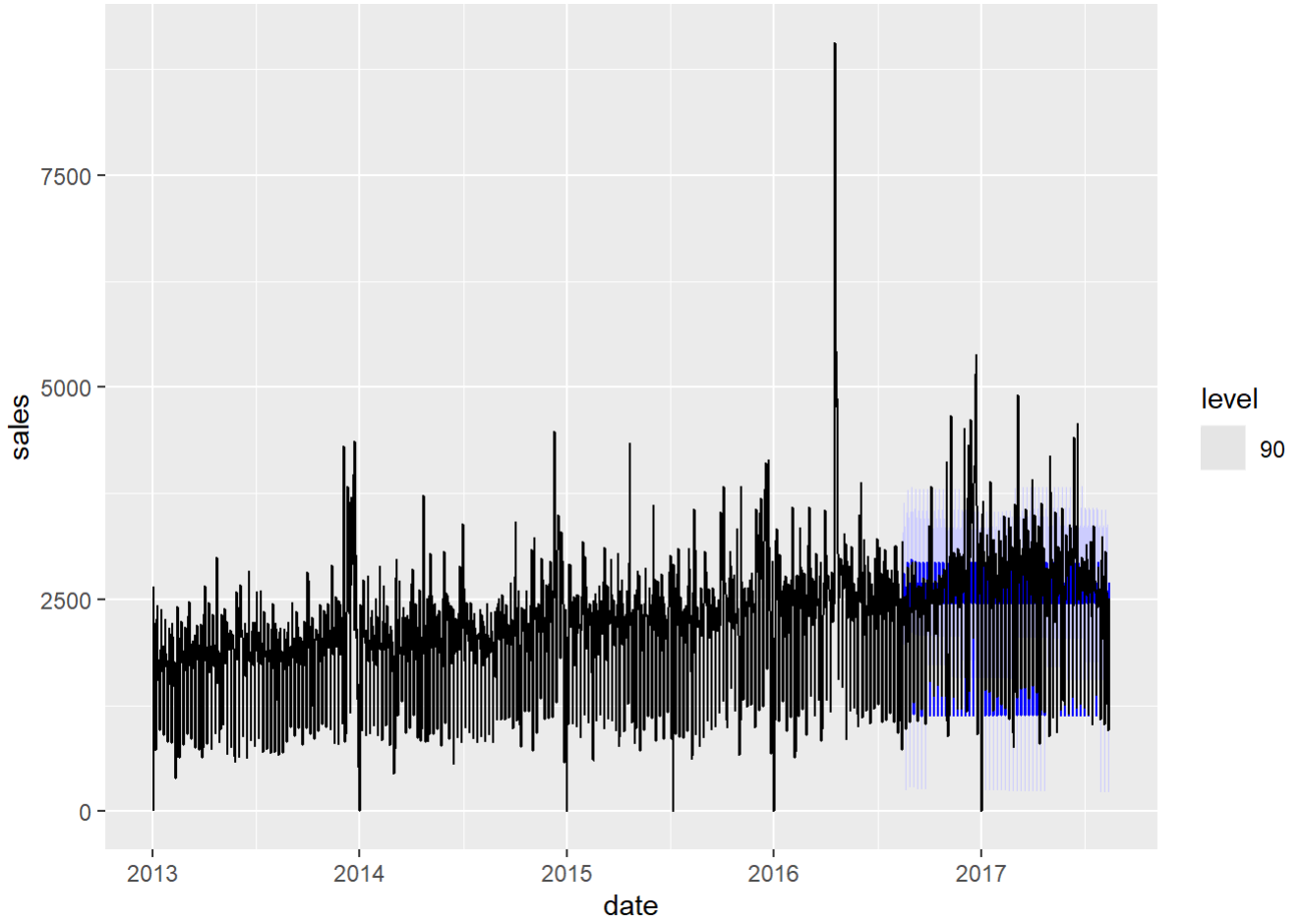
```
# fetching the forecast values for both the model fits
fc1 <- fit[1] %>% forecast(h = "1 year")
fc2 <- fit[2] %>% forecast(h = "1 year")
# plotting the forecast for auto arima
fc1 %>% autoplot(df_sel,level = 90)
```



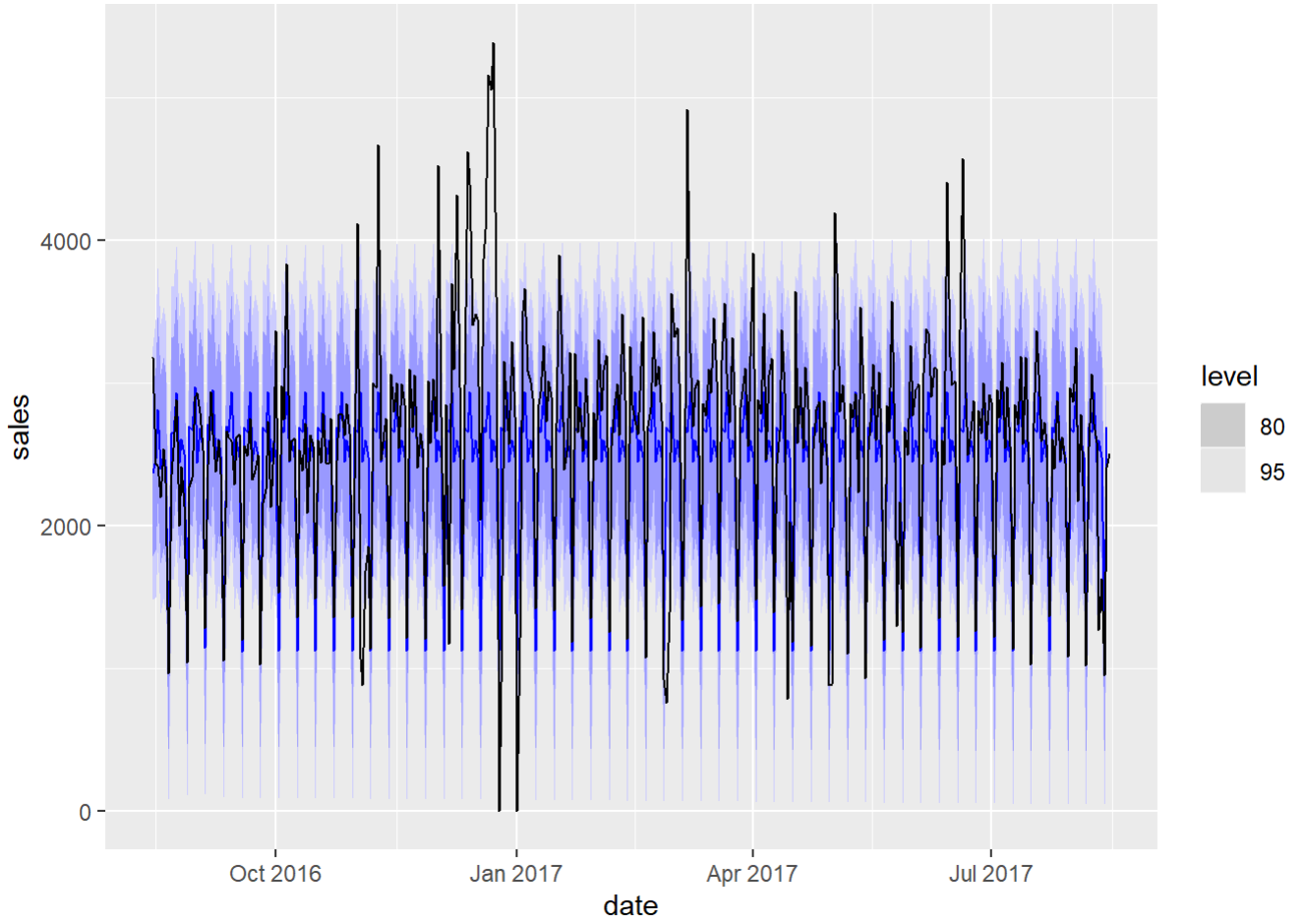

```
fc1 %>% autoplot(df_sel_test)
```



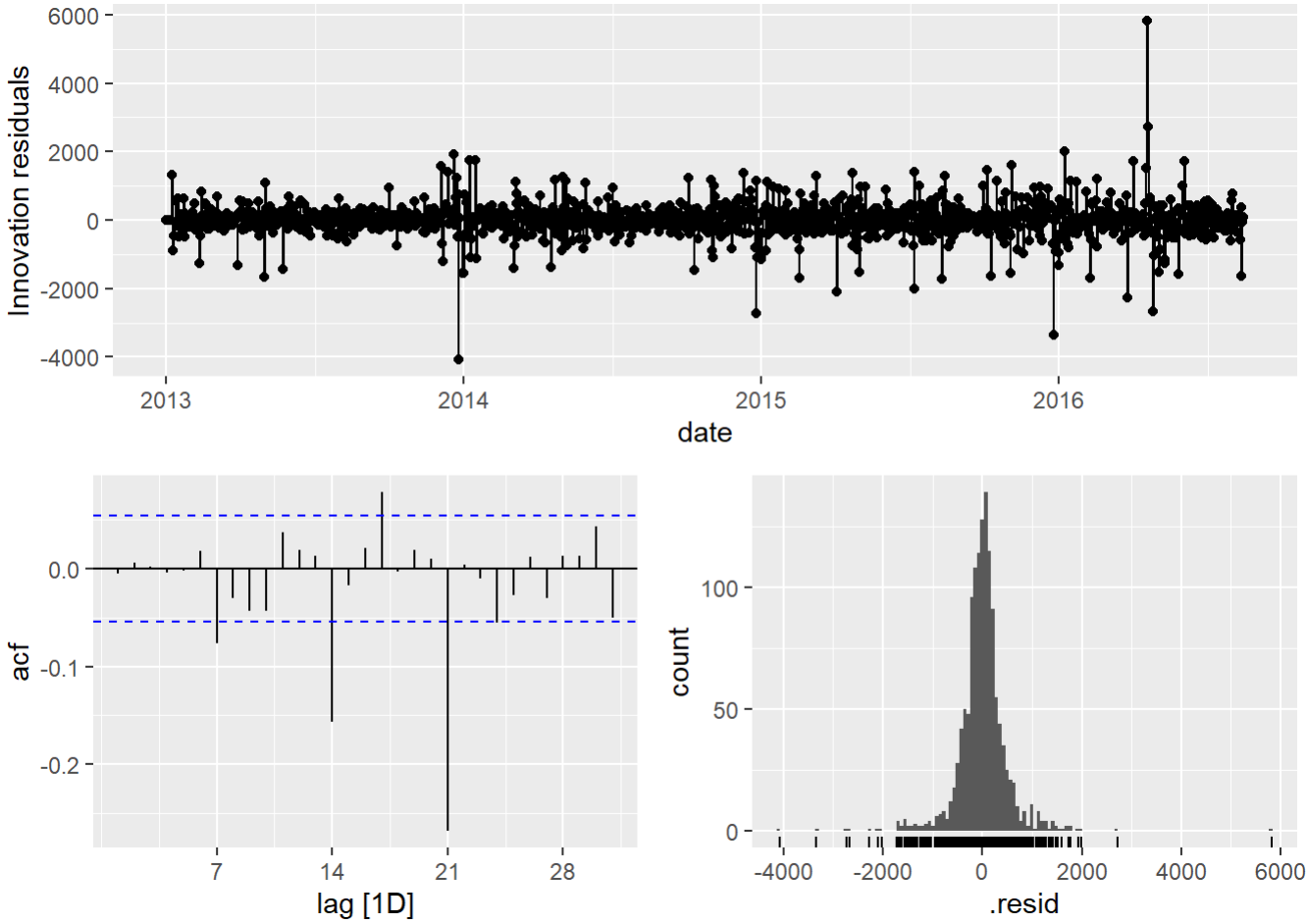

```
# plotting forecasted values of chosen ARIMA model
fc2 %>% autoplot(df_sel,level = 90)
```



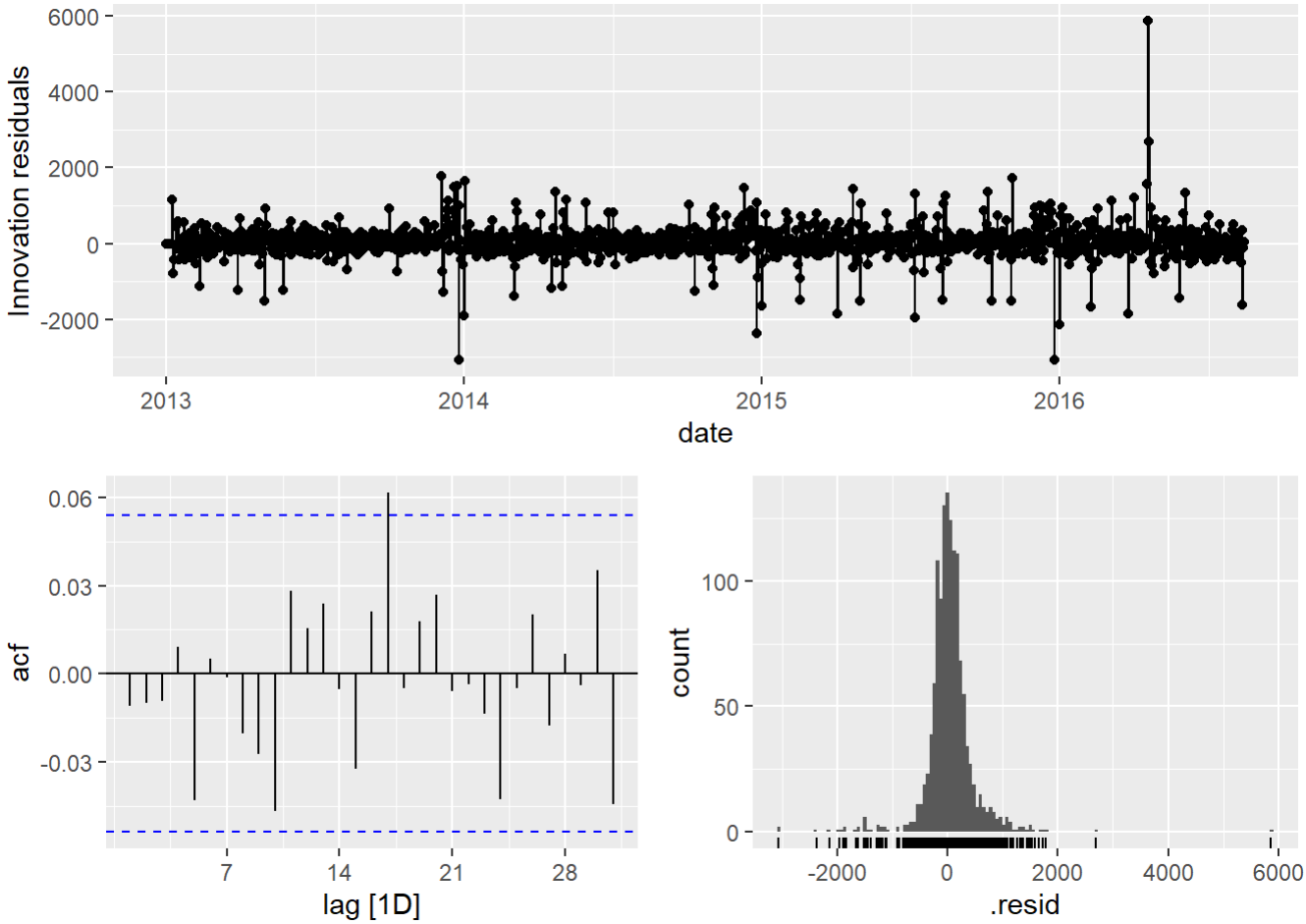

```
fc2 %>% autoplot(df_sel_test)
```




```
# performing residual test for both models
gg_tsresiduals(fit[1])
```




```
gg_tsresiduals(fit[2])
```



From ACF plot, we can see a significant spike at lag = 21 for auto ARIMA model. For our chosen ARIMA model, there is only a small spike in ACF plot at lag = 17, which could be possible white noise.

```
# fetching accuracy values for forecasted data.
accuracy(fc1,df_sel)
```

.model	.type	ME	RMSE	MAE	M...	M...	MASE	RMSSE	ACF1
<chr>	<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
arima_auto	Test	485.6574	814.1161	605.448	-Inf	Inf	1.524154	1.165817	0.2844194

1 row

```
accuracy(fc2,df_sel)
```

.model	.type	ME	RMSE	MAE	M...	M...	MASE	RMSSE	ACF1
<chr>	<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
arima1	Test	187.6929	603.3284	392.1091	-Inf	Inf	0.9870948	0.8639686	0.3724593

1 row

RMSE values for chosen ARIMA model is less compared to auto ARIMA model. Hence, the chosen ARIMA model is producing the best performance so far for our dataset.

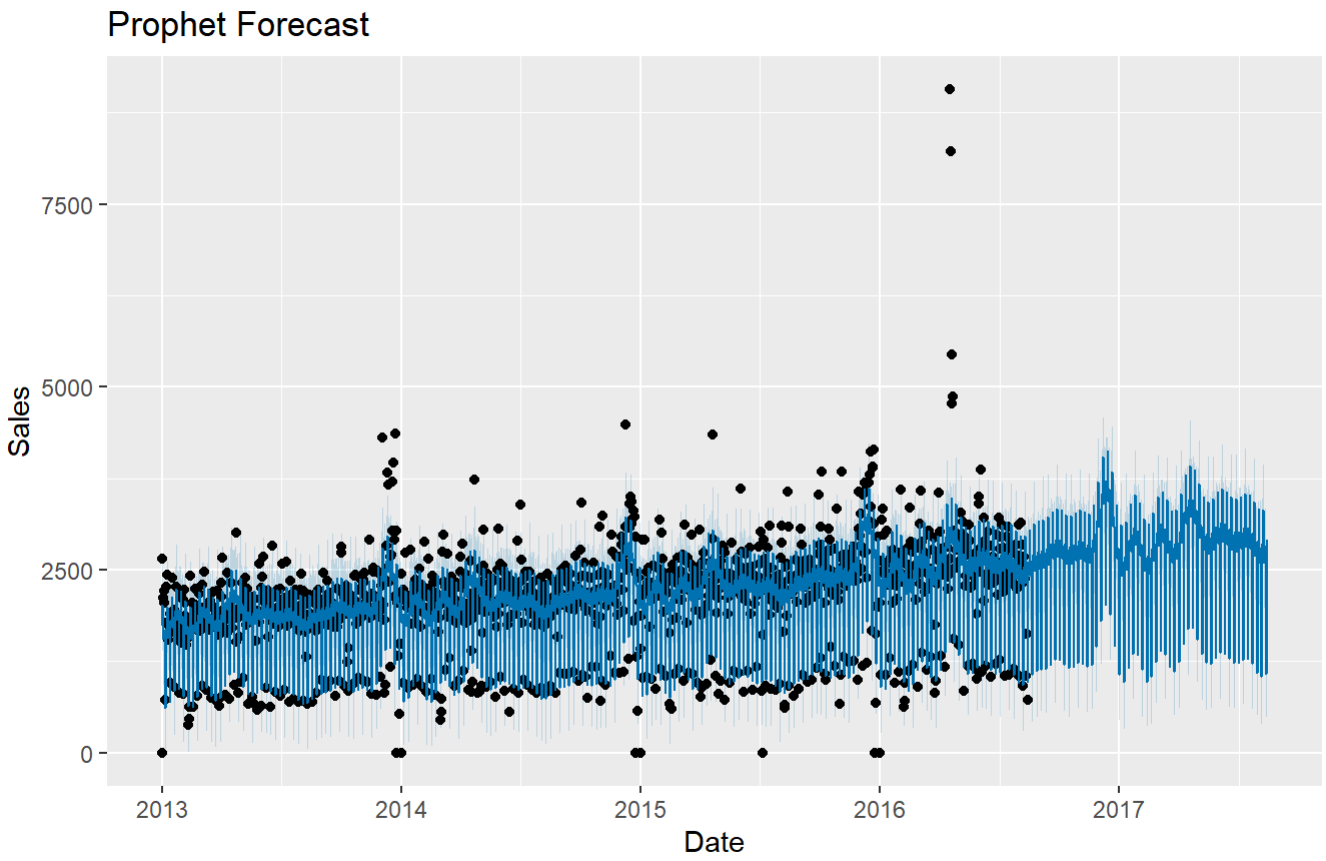
Prophet model

We are trying Prophet model due to its flexibility and automatic identification capabilities of trend, seasonality and holiday effects. It is also robust to outliers and missing data.

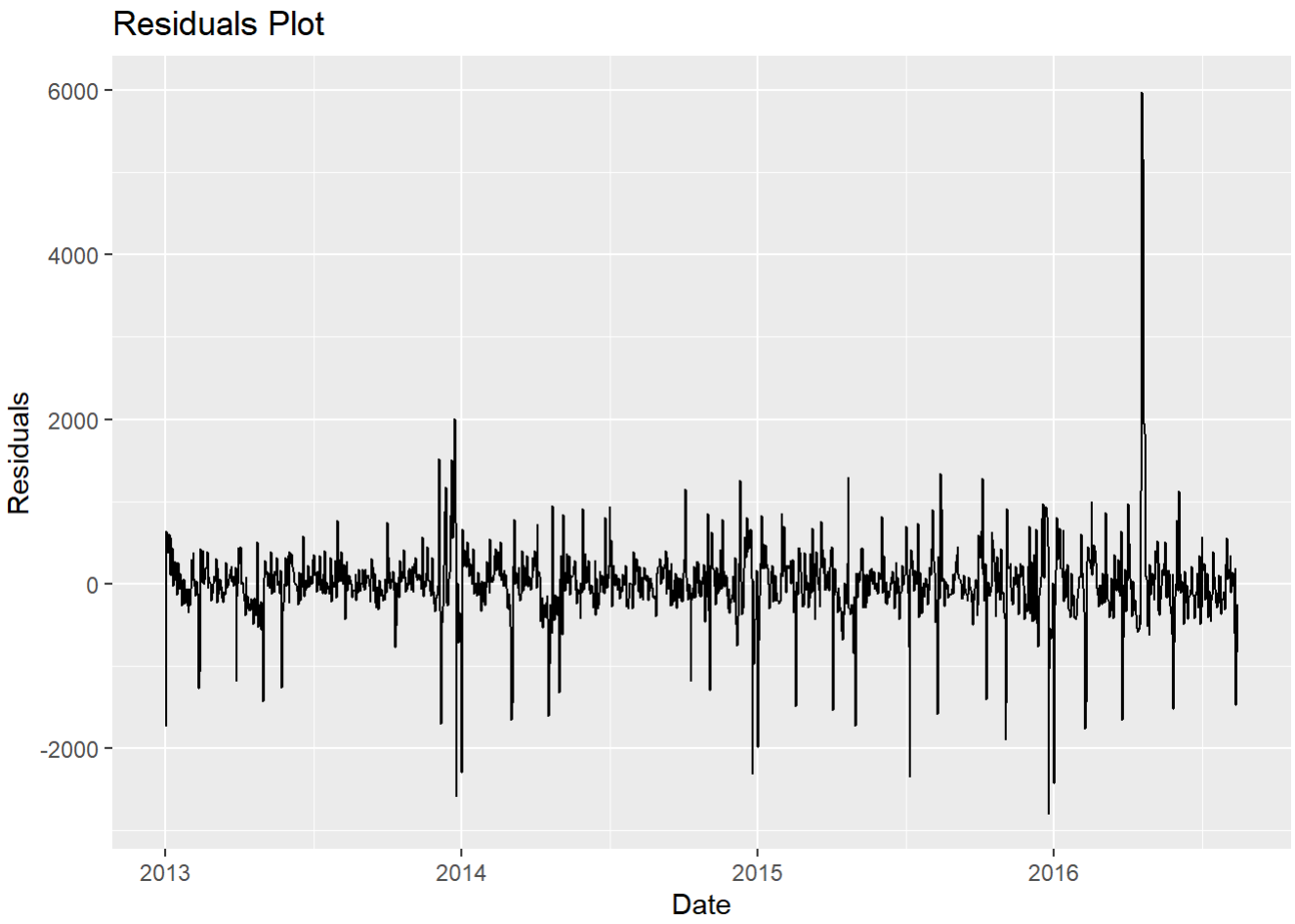
```
colnames(df_sel_train) <- c('ds', 'y')
m <- prophet(df_sel_train, seasonality.mode = 'multiplicative')
```

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

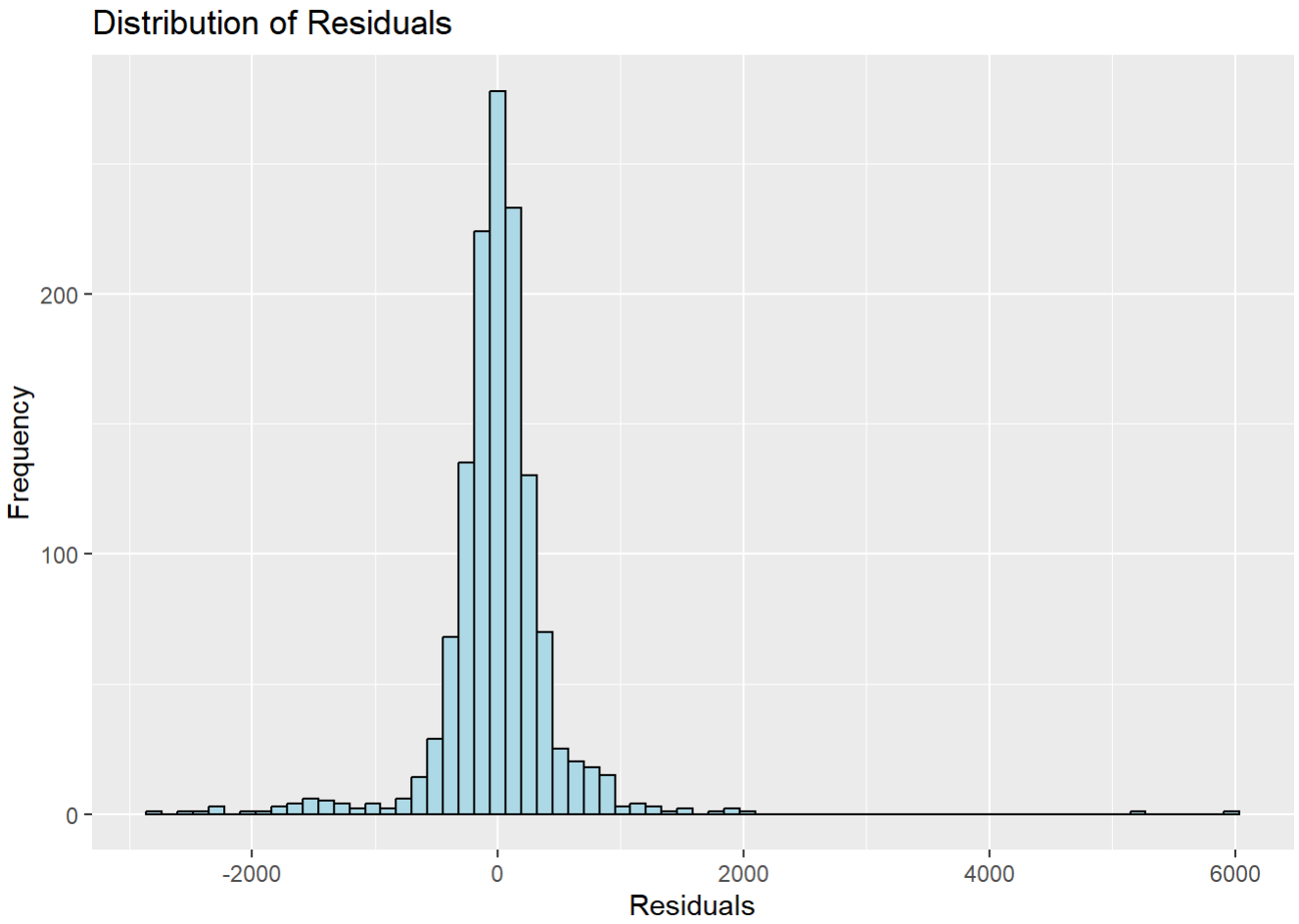
```
future <- make_future_dataframe(m, periods = 365, freq="day", include_history = TRUE)
forecast <- predict(m, future)
p <- plot(m,forecast)
p <- p + labs(x = "Date", y = "Sales")
p <- p + ggtitle("Prophet Forecast")
p
```



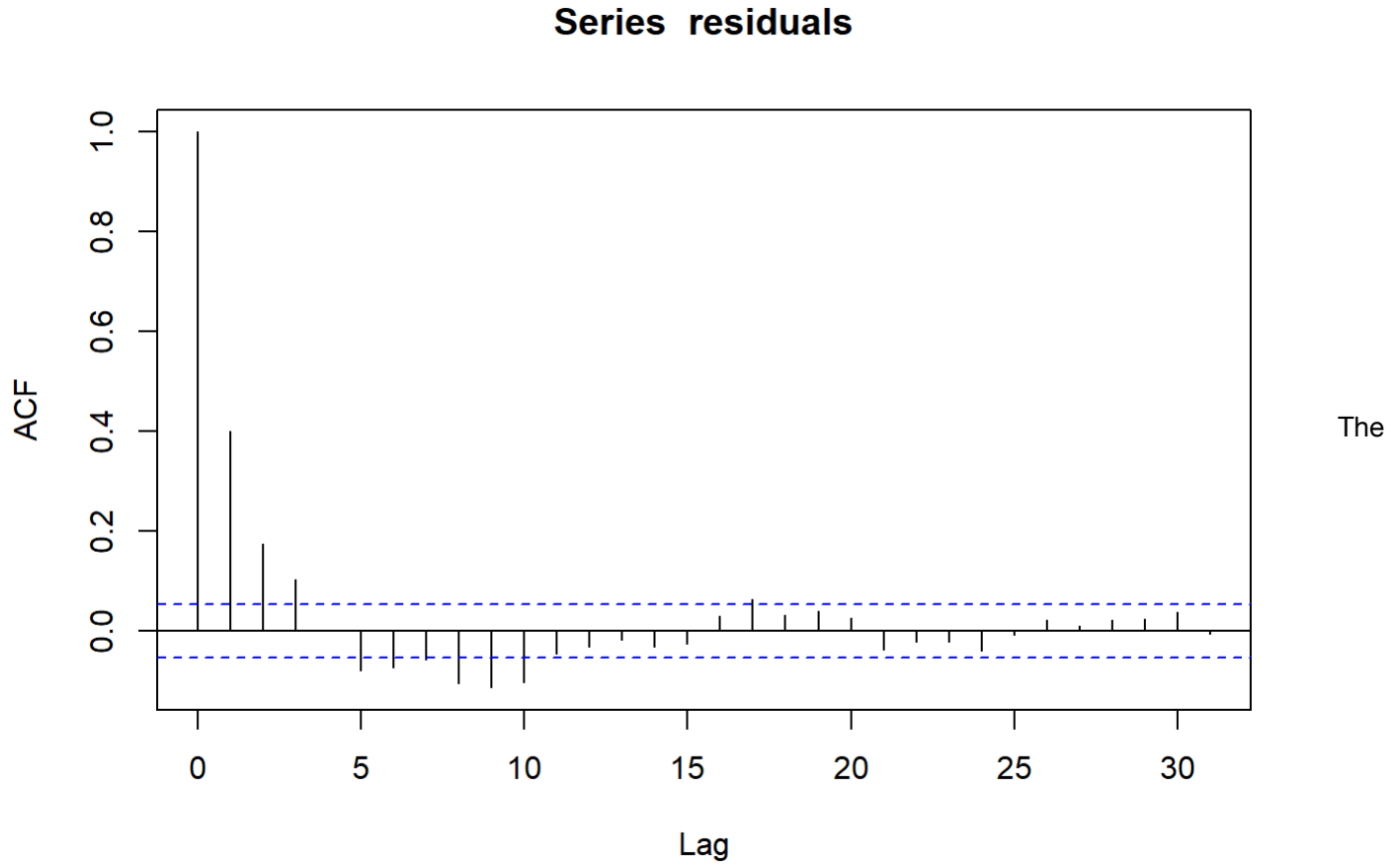
```
# residual test for prophet model
residuals <- df_sel_train$y - forecast$yhat[1:nrow(df_sel_train)]

ggplot(data.frame(ds = df_sel_train$ds, residuals = residuals), aes(x = ds, y = residual
s)) +
  geom_line() +
  labs(x = "Date", y = "Residuals") +
  ggtitle("Residuals Plot")
```




```
ggplot(data.frame(residuals = residuals), aes(x = residuals)) +
  geom_histogram(bins = 70, fill = "lightblue", color = "black") +
  labs(x = "Residuals", y = "Frequency") +
  ggtitle("Distribution of Residuals")
```




```
acf(residuals)
```



The residuals are normally distributed, and the ACF plot seems to be sinusoidally decreasing to zero.

```
library(forecast)
```

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

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

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

```
actual_values <- df_sel_train$y

forecasted_values <- forecast$yhat[1:length(actual_values)]

accuracy_metrics <- accuracy(forecasted_values, actual_values)

print(accuracy_metrics)
```

```
##                 ME     RMSE      MAE  MPE MAPE
## Test set 0.2315745 474.0274 269.9735 -Inf  Inf
```

Prophet model has given us Test RMSE values as 478, which is the lowest so far compared to ARIMA models, ETS models and Seasonal Naive model.

Project Conclusion:

Based on our comprehensive analysis and comparison of the forecasting models, the Prophet model emerged as the most effective in accurately predicting retail store sales. Leveraging advanced techniques for time series forecasting, the Prophet model incorporates trend and seasonality components while considering relevant external factors.

Additionally, we conducted a thorough examination of the residuals to evaluate the quality of the forecasting models. By visualizing the residuals and performing statistical analyses, we assessed the models' ability to capture and explain the remaining variation in the sales data.

In conclusion, this empirical study demonstrates the practical implementation of time series forecasting methods on a large-scale retail store sales dataset. The insights gained from our analysis based on our dataset, hold significant value for retail businesses seeking to optimize their sales strategies. The superior performance of the Prophet model highlights its potential for accurate sales forecasting, enabling businesses to make informed decisions and effectively manage their operations.