Unravelling Urban Traffic Dynamics: A Time-Series Analysis

Claire Bentzen, Conor Fitzpatrick, Nishok Narayanan

2023-11-15

library(ggplot2)  
library(corrplot)  
library(dplyr)  
library(lubridate)  
library(forecast)  
library(gridExtra)  
library(tidyr)  
library(knitr)

## Introduction

# read in traffic data  
traffic <- read.csv("traffic.csv")  
head(traffic)

## DateTime Junction Vehicles ID  
## 1 2015-11-01 00:00:00 1 15 20151101001  
## 2 2015-11-01 01:00:00 1 13 20151101011  
## 3 2015-11-01 02:00:00 1 10 20151101021  
## 4 2015-11-01 03:00:00 1 7 20151101031  
## 5 2015-11-01 04:00:00 1 9 20151101041  
## 6 2015-11-01 05:00:00 1 6 20151101051

# convert date column to datetime  
traffic$DateTime <- as.POSIXct(traffic$DateTime, format="%Y-%m-%d %H:%M:%S")  
  
# convert Junction column to categorical  
traffic$Junction <- factor(traffic$Junction)

# check for missing values by column  
colSums(is.na(traffic))

## DateTime Junction Vehicles ID   
## 7 0 0 0

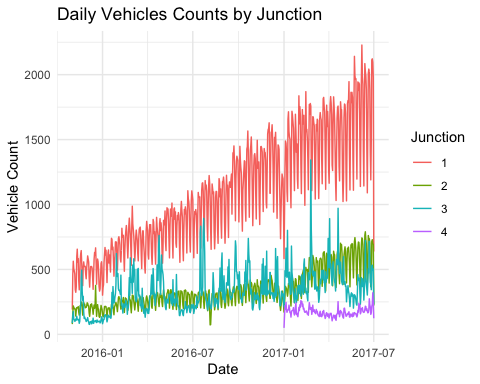
# remove NAs from dataframe  
traffic <- na.omit(traffic)  
  
# check for duplicate rows  
sum(duplicated(traffic))

## [1] 0

## Exploratory Data Analysis

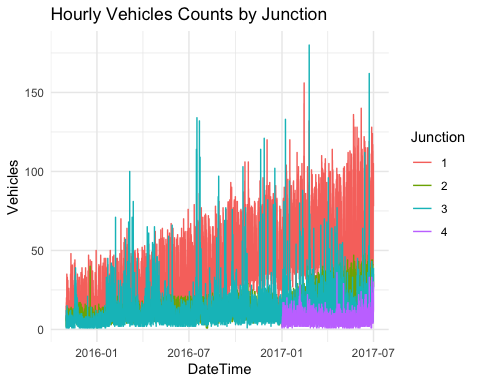
### Explore Vehicle Counts Aggregated by Day

# aggregate by date  
traffic$Date <- as.Date(traffic$DateTime)  
  
# group by Date and Junction, then calculate daily counts  
traffic\_daily <- traffic %>%  
 group\_by(Date, Junction) %>%  
 summarize(DailyCount = sum(Vehicles))  
  
# plot daily vehicle counts  
ggplot(traffic\_daily, aes(x = Date, y = DailyCount, color = Junction)) +  
 geom\_line() +  
 labs(title = "Daily Vehicles Counts by Junction",  
 x = "Date",  
 y = "Vehicle Count",  
 color = "Junction") +  
 theme\_minimal()



### Explore Vehicle Counts by Hour

# plot hourly vehicle counts  
ggplot(traffic, aes(x = DateTime, y = Vehicles, color = Junction)) +  
 geom\_line() +  
 labs(title = "Hourly Vehicles Counts by Junction",  
 x = "DateTime",  
 y = "Vehicles",  
 color = "Junction") +  
 theme\_minimal()

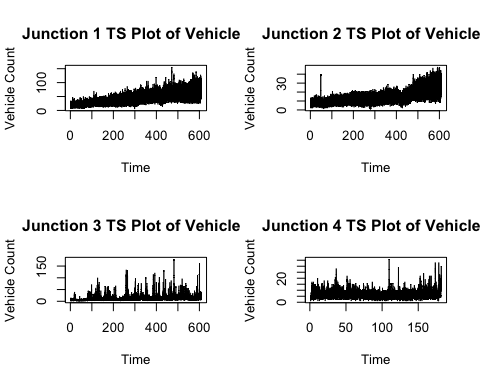


# average vehicle counts grouped by Junction  
vehicle\_avg <- traffic %>%  
 group\_by(Junction) %>%  
 summarize(AvgVehicleCount = mean(Vehicles))  
vehicle\_avg

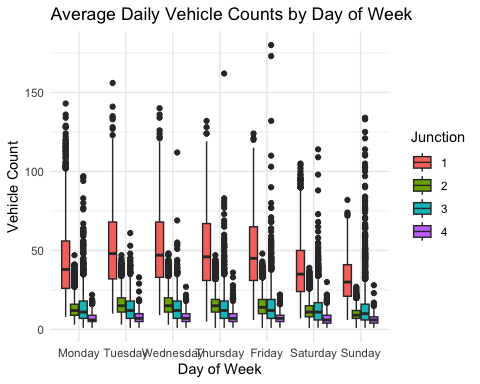
## # A tibble: 4 × 2  
## Junction AvgVehicleCount  
## <fct> <dbl>  
## 1 1 45.1   
## 2 2 14.3   
## 3 3 13.7   
## 4 4 7.25

### Time Series Plot by Junction

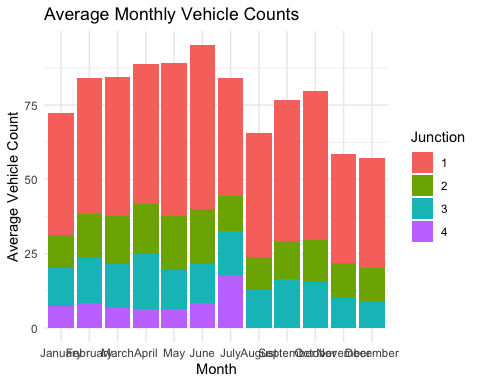
# split data into each junction  
traffic1 <- traffic[traffic$Junction == 1, ]  
traffic2 <- traffic[traffic$Junction == 2, ]  
traffic3 <- traffic[traffic$Junction == 3, ]  
traffic4 <- traffic[traffic$Junction == 4, ]  
  
# plot each junction vehicle counts  
par(mfrow = c(2, 2))  
  
# junction 1 time series  
traffic1.ts <- ts(traffic1$Vehicles, frequency = 24)  
plot.ts(traffic1.ts, xlab = "Time", ylab = "Vehicle Count", main = "Junction 1 TS Plot of Vehicles")   
  
# junction 2 time series  
traffic2.ts <- ts(traffic2$Vehicles, frequency = 24)  
plot.ts(traffic2.ts, xlab = "Time", ylab = "Vehicle Count", main = "Junction 2 TS Plot of Vehicles")   
  
# junction 3 time series  
traffic3.ts <- ts(traffic3$Vehicles, frequency = 24)  
plot.ts(traffic3.ts, xlab = "Time", ylab = "Vehicle Count", main = "Junction 3 TS Plot of Vehicles")   
  
# junction 4 time series  
traffic4.ts <- ts(traffic4$Vehicles, frequency = 24)  
plot.ts(traffic4.ts, xlab = "Time", ylab = "Vehicle Count", main = "Junction 4 TS Plot of Vehicles")



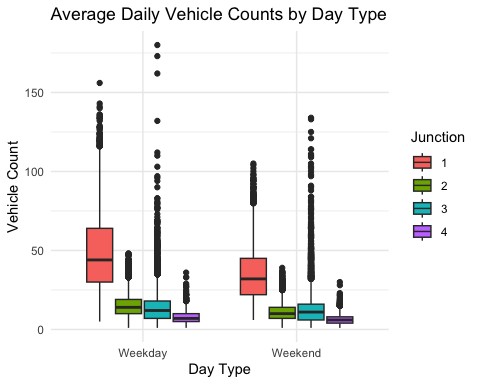
# Convert Date column to Date object  
traffic$Date <- as.Date(traffic$Date)  
  
# Extract day of week from Date  
traffic$DayOfWeek <- weekdays(traffic$Date)  
  
# Order days  
day\_order <- c("Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday")  
traffic$DayOfWeek <- factor(traffic$DayOfWeek, levels = day\_order)  
  
# Plot average daily vehicle counts by day of week  
ggplot(traffic, aes(x = DayOfWeek, y = Vehicles, fill = Junction)) +  
 geom\_boxplot() +  
 labs(title = "Average Daily Vehicle Counts by Day of Week",  
 x = "Day of Week",  
 y = "Vehicle Count",  
 fill = "Junction") +  
 theme\_minimal()



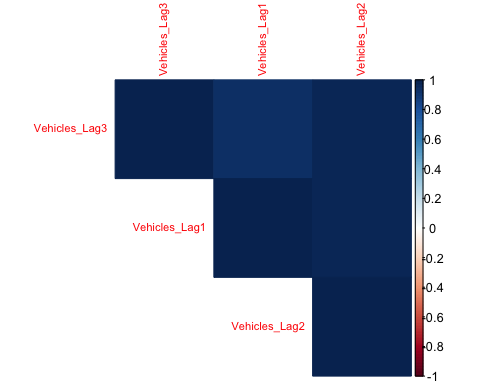
# Extract month from Date  
traffic$Month <- format(traffic$Date, "%B")  
  
# Order months  
month\_order <- c("January", "February", "March", "April", "May", "June", "July", "August", "September", "October", "November", "December")  
traffic$Month <- factor(traffic$Month, levels = month\_order)  
  
# Plot average monthly vehicle counts  
ggplot(traffic, aes(x = Month, y = Vehicles, fill = Junction)) +  
 geom\_bar(stat = "summary", fun = "mean") +  
 labs(title = "Average Monthly Vehicle Counts",  
 x = "Month",  
 y = "Average Vehicle Count",  
 fill = "Junction") +  
 theme\_minimal()



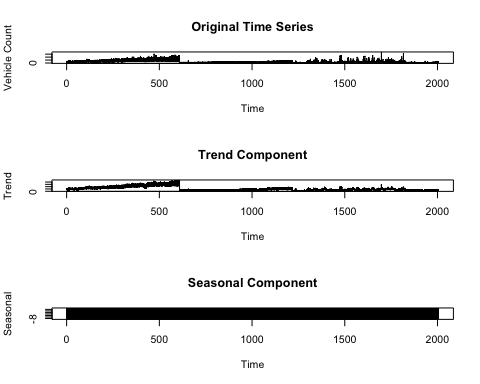
# Create a binary variable indicating weekday or weekend  
traffic$DayType <- ifelse(weekdays(traffic$Date) %in% c("Saturday", "Sunday"), "Weekend", "Weekday")  
  
# Plot average daily vehicle counts by day type  
ggplot(traffic, aes(x = DayType, y = Vehicles, fill = Junction)) +  
 geom\_boxplot() +  
 labs(title = "Average Daily Vehicle Counts by Day Type",  
 x = "Day Type",  
 y = "Vehicle Count",  
 fill = "Junction") +  
 theme\_minimal()



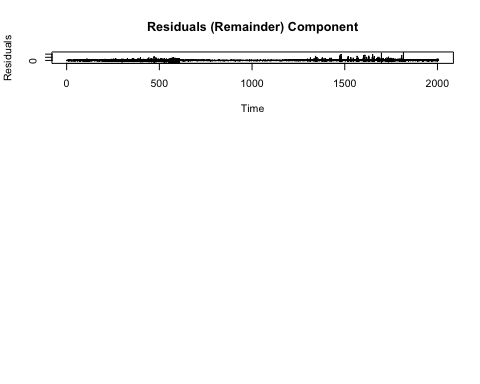
# Convert DateTime to POSIXct  
traffic$DateTime <- as.POSIXct(traffic$DateTime, format="%Y-%m-%d %H:%M:%S")  
  
# Calculate lagged variables for each Junction  
traffic\_lagged <- traffic %>%  
 group\_by(Junction) %>%  
 arrange(DateTime) %>%  
 mutate(Vehicles\_Lag1 = lag(Vehicles),  
 Vehicles\_Lag2 = lag(Vehicles, 2),  
 Vehicles\_Lag3 = lag(Vehicles, 3))  
  
# Filter out rows with missing lagged values  
traffic\_lagged <- filter(traffic\_lagged, complete.cases(Vehicles, Vehicles\_Lag1, Vehicles\_Lag2, Vehicles\_Lag3))  
  
# Ensure 'traffic\_lagged' is a data frame  
traffic\_lagged <- as.data.frame(traffic\_lagged)  
  
# Check for infinite values  
if (any(is.infinite(unlist(traffic\_lagged)))) {  
 stop("Infinite values detected in the data.")  
}  
  
# Calculate correlation matrix for the lagged vehicle counts of different Junctions  
cor\_matrix\_junctions <- cor(traffic\_lagged %>%  
 select(starts\_with("Vehicles\_Lag")), use = "pairwise.complete.obs")  
  
# Plot the correlation matrix as a heatmap  
corrplot(cor\_matrix\_junctions, method = "color", type = "upper", order = "hclust", tl.cex = 0.7)



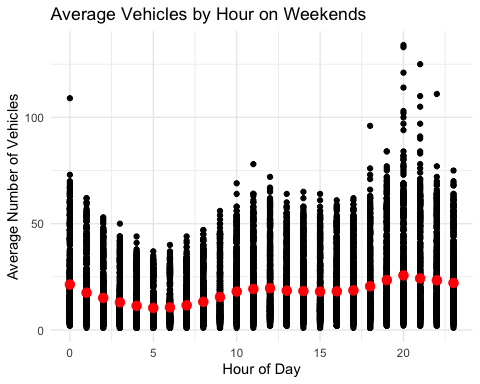
# Convert DateTime to POSIXct  
traffic$DateTime <- as.POSIXct(traffic$DateTime, format="%Y-%m-%d %H:%M:%S")  
  
# Create a time series object  
traffic\_ts <- ts(traffic$Vehicles, frequency = 24)  
  
# Time Series Decomposition  
decomposition <- decompose(traffic\_ts)  
  
# Plot the original time series  
par(mfrow = c(3, 1))  
plot(traffic\_ts, main = "Original Time Series", ylab = "Vehicle Count")  
  
# Plot the trend component  
plot(decomposition$trend, main = "Trend Component", ylab = "Trend")  
  
# Plot the seasonal component  
plot(decomposition$seasonal, main = "Seasonal Component", ylab = "Seasonal")



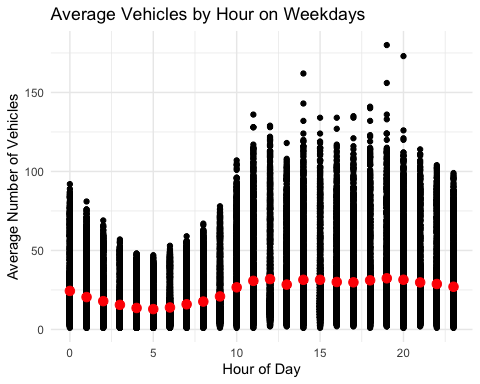
# Plot the remainder (residuals) component  
plot(decomposition$random, main = "Residuals (Remainder) Component", ylab = "Residuals")



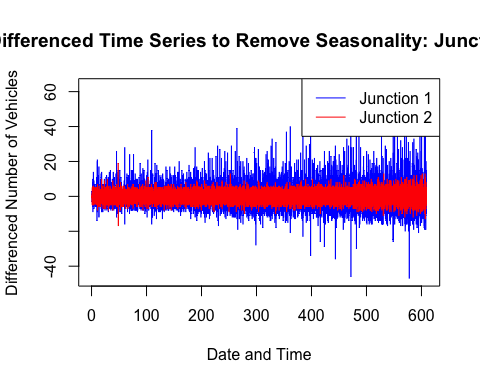
# Look at hour of day for weekends   
  
# Convert DateTime to a datetime object  
traffic$DateTime <- as.POSIXct(traffic$DateTime, format="%Y-%m-%d %H:%M:%S")  
  
# Create a new column for the hour of the day  
traffic$HourOfDay <- format(traffic$DateTime, "%H")  
  
# Convert HourOfDay to numeric for plotting purposes  
traffic$HourOfDay <- as.numeric(traffic$HourOfDay)  
  
# Create a new column for the day of the week  
traffic$DayOfWeek <- weekdays(traffic$DateTime)  
  
# Filter data to include only weekends  
traffic\_weekends <- traffic[traffic$DayOfWeek %in% c("Saturday", "Sunday"), ]  
  
# Plot the average number of vehicles for each hour on weekends  
ggplot(traffic\_weekends, aes(x = HourOfDay, y = Vehicles)) +  
 geom\_point() +  
 stat\_summary(fun.y = "mean", geom = "point", col = "red", size = 3) +  
 labs(x = "Hour of Day", y = "Average Number of Vehicles", title = "Average Vehicles by Hour on Weekends") +  
 theme\_minimal()



# Weekdays hour of day  
  
# Convert DateTime to a datetime object  
traffic$DateTime <- as.POSIXct(traffic$DateTime, format="%Y-%m-%d %H:%M:%S")  
  
# Create a new column for the hour of the day  
traffic$HourOfDay <- format(traffic$DateTime, "%H")  
  
# Convert HourOfDay to numeric for plotting purposes  
traffic$HourOfDay <- as.numeric(traffic$HourOfDay)  
  
# Create a new column for the day of the week  
traffic$DayOfWeek <- weekdays(traffic$DateTime)  
  
# Filter data to include only weekdays  
traffic\_weekdays <- traffic[traffic$DayOfWeek %in% c("Monday", "Tuesday", "Wednesday", "Thursday", "Friday"), ]  
  
# Plot the average number of vehicles for each hour on weekdays  
ggplot(traffic\_weekdays, aes(x = HourOfDay, y = Vehicles)) +  
 geom\_point() +  
 stat\_summary(fun.y = "mean", geom = "point", col = "red", size = 3) +  
 labs(x = "Hour of Day", y = "Average Number of Vehicles", title = "Average Vehicles by Hour on Weekdays") +  
 theme\_minimal()



# Differencing   
  
traffic$DateTime <- as.POSIXct(traffic$DateTime)  
  
# Create a time series object for junctions 1 and 2  
ts\_junction1 <- ts(traffic$Vehicles[traffic$Junction == 1], frequency = 24) # Assuming hourly data  
ts\_junction2 <- ts(traffic$Vehicles[traffic$Junction == 2], frequency = 24) # Assuming hourly data  
  
# Differencing to remove seasonality  
diff\_junction1 <- diff(ts\_junction1)  
diff\_junction2 <- diff(ts\_junction2)  
  
# Create a new time series with DateTime  
ts\_diff\_junction1 <- ts(diff\_junction1, start = start(ts\_junction1), frequency = frequency(ts\_junction1))  
ts\_diff\_junction2 <- ts(diff\_junction2, start = start(ts\_junction2), frequency = frequency(ts\_junction2))  
  
# Plot the differenced time series for Junctions 1 and 2  
plot.new()  
plot(ts\_diff\_junction1, type = "l", col = "blue",  
 xlab = "Date and Time", ylab = "Differenced Number of Vehicles",  
 main = "Differenced Time Series to Remove Seasonality: Junction 1")  
lines(ts\_diff\_junction2, col = "red")  
legend("topright", legend = c("Junction 1", "Junction 2"), col = c("blue", "red"), lty = 1)



#Split into Junction 1,2,3 and perform log transformation  
  
traffic <- read.csv("traffic.csv")  
traffic <- traffic %>%  
 select(-ID)  
  
# Convert DateTime column to POSIXct  
traffic$DateTime <- as.POSIXct(traffic$DateTime)  
  
# Filter out Junction 4  
filtered\_traffic <- traffic %>%  
 filter(Junction != 4)  
  
# Split the data into Junctions 1, 2, and 3  
junction1\_data <- filtered\_traffic %>%  
 filter(Junction == 1)  
  
junction2\_data <- filtered\_traffic %>%  
 filter(Junction == 2)  
  
junction3\_data <- filtered\_traffic %>%  
 filter(Junction == 3)  
  
  
# Function to perform log transformation on the "Vehicles" column  
log\_transformation\_function <- function(data) {  
 data$Vehicles\_log <- log1p(data$Vehicles)  
 return(data)  
}  
  
# Apply log transformation to each junction dataset  
junction1\_data <- log\_transformation\_function(junction1\_data)  
junction2\_data <- log\_transformation\_function(junction2\_data)  
junction3\_data <- log\_transformation\_function(junction3\_data)  
  
# Print the updated data frames  
cat("\nJunction 1 Data with Log Transformation:\n")

##   
## Junction 1 Data with Log Transformation:

head(junction1\_data)

## DateTime Junction Vehicles Vehicles\_log  
## 1 2015-11-01 1 15 2.772589  
## 2 2015-11-01 1 13 2.639057  
## 3 2015-11-01 1 10 2.397895  
## 4 2015-11-01 1 7 2.079442  
## 5 2015-11-01 1 9 2.302585  
## 6 2015-11-01 1 6 1.945910

cat("\nJunction 2 Data with Log Transformation:\n")

##   
## Junction 2 Data with Log Transformation:

head(junction2\_data)

## DateTime Junction Vehicles Vehicles\_log  
## 1 2015-11-01 2 6 1.945910  
## 2 2015-11-01 2 6 1.945910  
## 3 2015-11-01 2 5 1.791759  
## 4 2015-11-01 2 6 1.945910  
## 5 2015-11-01 2 7 2.079442  
## 6 2015-11-01 2 2 1.098612

cat("\nJunction 3 Data with Log Transformation:\n")

##   
## Junction 3 Data with Log Transformation:

head(junction3\_data)

## DateTime Junction Vehicles Vehicles\_log  
## 1 2015-11-01 3 9 2.3025851  
## 2 2015-11-01 3 7 2.0794415  
## 3 2015-11-01 3 5 1.7917595  
## 4 2015-11-01 3 1 0.6931472  
## 5 2015-11-01 3 2 1.0986123  
## 6 2015-11-01 3 2 1.0986123

# Function to perform differencing on the "Vehicles" column  
difference\_function <- function(data) {  
 data$Vehicles\_diff <- c(NA, diff(data$Vehicles))  
 return(data)  
}  
  
# Apply differencing to each junction dataset  
junction1\_data <- difference\_function(junction1\_data)  
junction2\_data <- difference\_function(junction2\_data)  
junction3\_data <- difference\_function(junction3\_data)  
  
# Print the updated data frames  
cat("\nJunction 1 Data with Differencing:\n")

##   
## Junction 1 Data with Differencing:

head(junction1\_data)

## DateTime Junction Vehicles Vehicles\_log Vehicles\_diff  
## 1 2015-11-01 1 15 2.772589 NA  
## 2 2015-11-01 1 13 2.639057 -2  
## 3 2015-11-01 1 10 2.397895 -3  
## 4 2015-11-01 1 7 2.079442 -3  
## 5 2015-11-01 1 9 2.302585 2  
## 6 2015-11-01 1 6 1.945910 -3

cat("\nJunction 2 Data with Differencing:\n")

##   
## Junction 2 Data with Differencing:

head(junction2\_data)

## DateTime Junction Vehicles Vehicles\_log Vehicles\_diff  
## 1 2015-11-01 2 6 1.945910 NA  
## 2 2015-11-01 2 6 1.945910 0  
## 3 2015-11-01 2 5 1.791759 -1  
## 4 2015-11-01 2 6 1.945910 1  
## 5 2015-11-01 2 7 2.079442 1  
## 6 2015-11-01 2 2 1.098612 -5

cat("\nJunction 3 Data with Differencing:\n")

##   
## Junction 3 Data with Differencing:

head(junction3\_data)

## DateTime Junction Vehicles Vehicles\_log Vehicles\_diff  
## 1 2015-11-01 3 9 2.3025851 NA  
## 2 2015-11-01 3 7 2.0794415 -2  
## 3 2015-11-01 3 5 1.7917595 -2  
## 4 2015-11-01 3 1 0.6931472 -4  
## 5 2015-11-01 3 2 1.0986123 1  
## 6 2015-11-01 3 2 1.0986123 0

# train/test split  
train\_test\_split\_function <- function(data, train\_percentage = 0.8) {  
 split\_index <- round(nrow(data) \* train\_percentage)  
 train\_set <- data[1:split\_index, ]  
 test\_set <- data[(split\_index + 1):nrow(data), ]  
 return(list(train\_set = train\_set, test\_set = test\_set))  
}  
  
# Apply train/test split to each junction dataset  
split\_junction1 <- train\_test\_split\_function(junction1\_data)  
split\_junction2 <- train\_test\_split\_function(junction2\_data)  
split\_junction3 <- train\_test\_split\_function(junction3\_data)  
  
# Access the training and testing sets for each junction  
train\_junction1 <- split\_junction1$train\_set  
test\_junction1 <- split\_junction1$test\_set  
  
train\_junction2 <- split\_junction2$train\_set  
test\_junction2 <- split\_junction2$test\_set  
  
train\_junction3 <- split\_junction3$train\_set  
test\_junction3 <- split\_junction3$test\_set  
  
head(train\_junction1)

## DateTime Junction Vehicles Vehicles\_log Vehicles\_diff  
## 1 2015-11-01 1 15 2.772589 NA  
## 2 2015-11-01 1 13 2.639057 -2  
## 3 2015-11-01 1 10 2.397895 -3  
## 4 2015-11-01 1 7 2.079442 -3  
## 5 2015-11-01 1 9 2.302585 2  
## 6 2015-11-01 1 6 1.945910 -3

# Print the dimensions of the training and testing sets for each junction  
cat("\nJunction 1 Training Set Dimensions:\n")

##   
## Junction 1 Training Set Dimensions:

print(dim(train\_junction1))

## [1] 11674 5

cat("\nJunction 1 Testing Set Dimensions:\n")

##   
## Junction 1 Testing Set Dimensions:

print(dim(test\_junction1))

## [1] 2918 5

cat("\nJunction 2 Training Set Dimensions:\n")

##   
## Junction 2 Training Set Dimensions:

print(dim(train\_junction2))

## [1] 11674 5

cat("\nJunction 2 Testing Set Dimensions:\n")

##   
## Junction 2 Testing Set Dimensions:

print(dim(test\_junction2))

## [1] 2918 5

cat("\nJunction 3 Training Set Dimensions:\n")

##   
## Junction 3 Training Set Dimensions:

print(dim(train\_junction3))

## [1] 11674 5

cat("\nJunction 3 Testing Set Dimensions:\n")

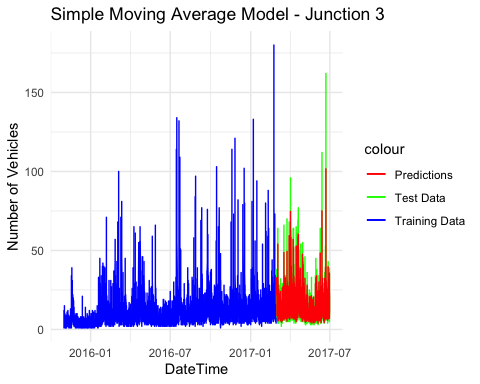
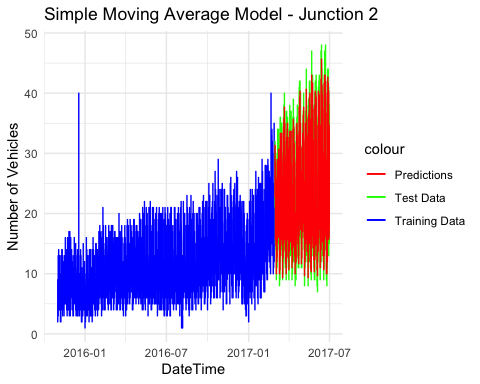
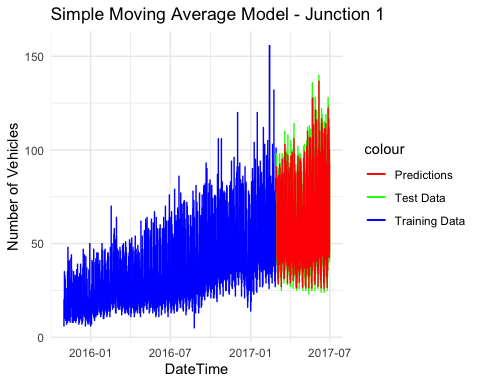
##   
## Junction 3 Testing Set Dimensions:

print(dim(test\_junction3))

## [1] 2918 5

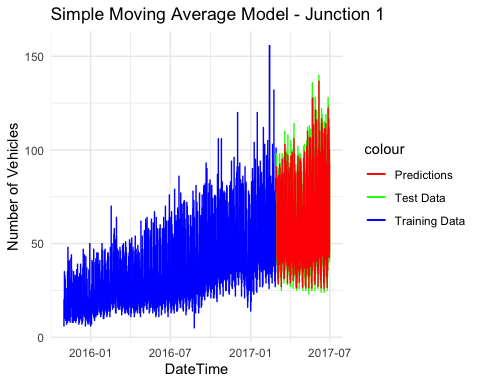
## Modeling

#MOVING AVERAGE MODEL  
  
# Function to calculate MAE, MSE, and RMSE  
calculate\_errors <- function(predictions, actual) {  
 errors <- predictions - actual  
 mae <- mean(abs(errors), na.rm = TRUE)  
 mse <- mean(errors^2, na.rm = TRUE)  
 rmse <- sqrt(mse)  
 return(c(MAE = mae, MSE = mse, RMSE = rmse))  
}  
  
# Function to calculate Simple Moving Average  
calculate\_sma <- function(data, window\_size) {  
 sma\_values <- zoo::rollmean(data, k = window\_size, fill = NA, align = "right")  
 return(sma\_values)  
}  
  
# Function to create plots for each junction  
create\_plots <- function(train\_data, test\_data, junction\_number, window\_size = 3) {  
 train\_data$SMA <- calculate\_sma(train\_data$Vehicles, window\_size)  
 test\_data$SMA <- calculate\_sma(test\_data$Vehicles, window\_size)  
   
 ggplot() +  
 geom\_line(data = train\_data, aes(x = DateTime, y = Vehicles, color = "Training Data")) +  
 geom\_line(data = test\_data, aes(x = DateTime, y = Vehicles, color = "Test Data")) +  
 geom\_line(data = test\_data, aes(x = DateTime, y = SMA, color = "Predictions")) +  
 labs(title = paste("Simple Moving Average Model - Junction", junction\_number),  
 x = "DateTime",  
 y = "Number of Vehicles") +  
 scale\_color\_manual(values = c("Training Data" = "blue", "Test Data" = "green", "Predictions" = "red")) +  
 theme\_minimal()  
}  
  
# Apply Simple Moving Average and create plots for each junction  
junctions <- list(junction1 = list(train\_data = train\_junction1, test\_data = test\_junction1),  
 junction2 = list(train\_data = train\_junction2, test\_data = test\_junction2),  
 junction3 = list(train\_data = train\_junction3, test\_data = test\_junction3))  
  
for (junction\_name in names(junctions)) {  
 plot\_data <- junctions[[junction\_name]]  
 plot <- create\_plots(plot\_data$train\_data, plot\_data$test\_data, junction\_number = substr(junction\_name, nchar(junction\_name), nchar(junction\_name)))  
 print(plot)  
}

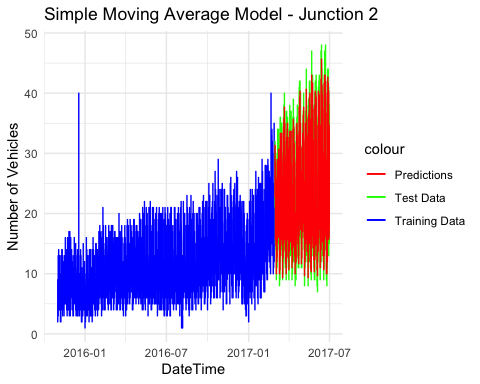


# Function to calculate MAE, MSE, and RMSE  
calculate\_errors <- function(predictions, actual) {  
 errors <- predictions - actual  
 mae <- mean(abs(errors), na.rm = TRUE)  
 mse <- mean(errors^2, na.rm = TRUE)  
 rmse <- sqrt(mse)  
 return(c(MAE = mae, MSE = mse, RMSE = rmse))  
}  
  
# Function to calculate Simple Moving Average  
calculate\_sma <- function(data, window\_size) {  
 sma\_values <- zoo::rollmean(data, k = window\_size, fill = NA, align = "right")  
 return(sma\_values)  
}  
  
# Initialize an empty dataframe to store errors  
total\_errors <- data.frame(Junction = character(0), MA = numeric(0), MSE = numeric(0), RMSE = numeric(0))  
  
# Apply Simple Moving Average and accumulate errors for each junction  
junctions <- list(junction1 = list(train\_data = train\_junction1, test\_data = test\_junction1),  
 junction2 = list(train\_data = train\_junction2, test\_data = test\_junction2),  
 junction3 = list(train\_data = train\_junction3, test\_data = test\_junction3))  
  
for (junction\_name in names(junctions)) {  
 plot\_data <- junctions[[junction\_name]]  
   
 # Calculate SMA for test data using Vehicles\_log  
 plot\_data$test\_data$SMA <- calculate\_sma(plot\_data$test\_data$Vehicles\_log, window\_size=3)  
   
 # Calculate errors for the current junction using Vehicles\_log  
 errors <- calculate\_errors(plot\_data$test\_data$SMA, plot\_data$test\_data$Vehicles\_log)  
   
 # Add junction name to errors  
 errors <- cbind(Junction = junction\_name, errors)  
   
 # Accumulate errors in the total\_errors dataframe  
 total\_errors <- rbind(total\_errors, data.frame(errors))  
   
 # Print errors for the current junction  
 cat("Junction", junction\_name, "Errors:\n")  
 print(errors)  
   
 # Create plots using Vehicles\_log  
 plot <- create\_plots(plot\_data$train\_data, plot\_data$test\_data, junction\_number = substr(junction\_name, nchar(junction\_name), nchar(junction\_name)))  
 print(plot)  
}

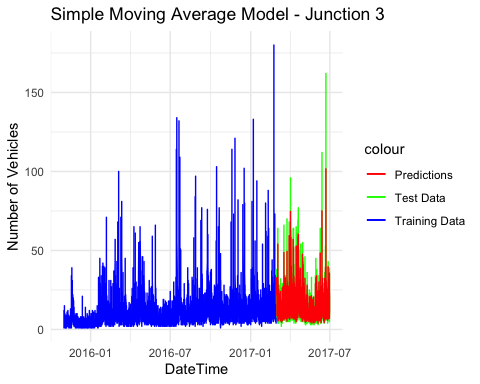
## Junction junction1 Errors:  
## Junction errors   
## MAE "junction1" "0.0859616664431755"  
## MSE "junction1" "0.0115803647435144"  
## RMSE "junction1" "0.107612103146042"



## Junction junction2 Errors:  
## Junction errors   
## MAE "junction2" "0.0896512990198146"  
## MSE "junction2" "0.0128518459009931"  
## RMSE "junction2" "0.113365982115417"



## Junction junction3 Errors:  
## Junction errors   
## MAE "junction3" "0.152297554342819"  
## MSE "junction3" "0.038815545420992"  
## RMSE "junction3" "0.197016612043229"



# Print the final dataframe with errors for all junctions  
print("Total Errors:")

## [1] "Total Errors:"

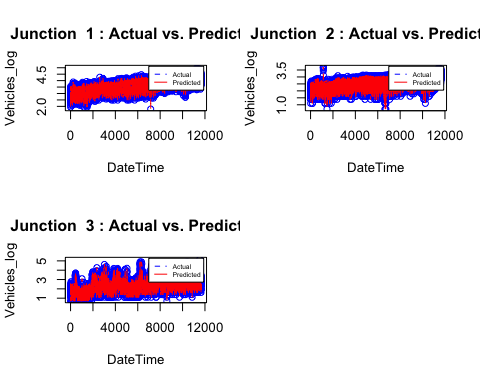
print(total\_errors)

## Junction errors  
## MAE junction1 0.0859616664431755  
## MSE junction1 0.0115803647435144  
## RMSE junction1 0.107612103146042  
## MAE1 junction2 0.0896512990198146  
## MSE1 junction2 0.0128518459009931  
## RMSE1 junction2 0.113365982115417  
## MAE2 junction3 0.152297554342819  
## MSE2 junction3 0.038815545420992  
## RMSE2 junction3 0.197016612043229

total\_errors

## Junction errors  
## MAE junction1 0.0859616664431755  
## MSE junction1 0.0115803647435144  
## RMSE junction1 0.107612103146042  
## MAE1 junction2 0.0896512990198146  
## MSE1 junction2 0.0128518459009931  
## RMSE1 junction2 0.113365982115417  
## MAE2 junction3 0.152297554342819  
## MSE2 junction3 0.038815545420992  
## RMSE2 junction3 0.197016612043229

# Seasonal Naive Model  
  
# Set the percentage of data for training (e.g., 80%)  
 train\_percentage <- 0.8  
  
# Initialize a list to store models and predictions for each junction  
junction\_models\_sn <- list()  
junction\_predictions\_sn <- list()  
  
# Define a function for fitting and forecasting  
fit\_sn\_model <- function(train\_data, test\_data) {  
 # Fit a Seasonal Naive model  
 sn\_model <- snaive(train\_data, h = length(test\_data))  
   
 # Make predictions on the test set  
 predictions\_sn <- forecast(sn\_model)  
   
 # Return the model and predictions  
 return(list(model = sn\_model, predictions = predictions\_sn))  
}  
  
# Initialize a list to store models and predictions for each junction  
junction\_models\_sn <- list()  
junction\_predictions\_sn <- list()  
  
# Define a function for fitting and forecasting  
fit\_sn\_model <- function(train\_data, test\_data) {  
 # Fit a Seasonal Naive model  
 sn\_model <- snaive(train\_data, h = length(test\_data))  
   
 # Make predictions on the test set  
 predictions\_sn <- forecast(sn\_model)  
   
 # Return the model and predictions  
 return(list(model = sn\_model, predictions = predictions\_sn))  
}  
  
# Iterate over each junction  
for (j in 1:3) {  
 # Get the train and test data for the current junction  
 train\_data <- eval(parse(text = paste0("train\_junction", j, "$Vehicles\_log")))  
 test\_data <- eval(parse(text = paste0("test\_junction", j, "$Vehicles\_log")))  
   
 # Fit the Seasonal Naive model  
 result\_junction <- fit\_sn\_model(train\_data, test\_data)  
   
 # Save the model and predictions in the lists  
 junction\_models\_sn[[as.character(j)]] <- result\_junction$model  
 junction\_predictions\_sn[[as.character(j)]] <- result\_junction$predictions  
}  
  
# Plotting  
par(mfrow = c(2, 2)) # 2x2 layout for the plots  
  
for (j in 1:3) {  
 # Get the data and predictions for the current junction  
 junction\_data <- eval(parse(text = paste0("train\_junction", j, "$Vehicles\_log")))  
 predictions\_sn <- junction\_predictions\_sn[[as.character(j)]]  
  
 # Plotting  
 plot(junction\_data, col = "blue", main = paste("Junction ", j, ": Actual vs. Predicted"), xlab = "DateTime", ylab = "Vehicles\_log")  
 lines(predictions\_sn$fitted, col = "red", lty = 2)  
 legend("topright", legend = c("Actual", "Predicted"), col = c("blue", "red"), lty = 2:1, cex = 0.5, inset = 0.02)  
}



# ARIMA Model  
# Function to calculate MAE, MSE, and RMSE  
calculate\_metrics <- function(actual, forecast) {  
 errors <- actual - forecast  
 mae <- mean(abs(errors), na.rm = TRUE)  
 mse <- mean(errors^2, na.rm = TRUE)  
 rmse <- sqrt(mse)  
   
 return(c(MAE = mae, MSE = mse, RMSE = rmse))  
}  
  
# ARIMA Model  
# Initialize lists to store models, predictions, and metrics for each junction  
junction\_models\_arima <- list()  
junction\_predictions\_arima <- list()  
junction\_metrics\_arima <- list()  
  
# Define a function for fitting and forecasting ARIMA  
fit\_arima\_model <- function(train\_data, test\_data) {  
 # Fit an ARIMA model  
 arima\_model <- Arima(train\_data, order = c(1, 1, 1))  
   
 # Make predictions on the test set  
 predictions\_arima <- forecast(arima\_model, h = length(test\_data))  
   
 # Calculate metrics  
 metrics <- calculate\_metrics(test\_data, predictions\_arima$mean)  
   
 # Return the model, predictions, and metrics  
 return(list(model = arima\_model, predictions = predictions\_arima, metrics = metrics))  
}  
  
# Initialize an empty dataframe to store metrics  
total\_metrics\_arima <- data.frame(Junction = character(0), MAE = numeric(0), MSE = numeric(0), RMSE = numeric(0))  
  
# Iterate over each junction  
for (j in 1:3) {  
 # Get the train and test data for the current junction  
 train\_data <- eval(parse(text = paste0("train\_junction", j, "$Vehicles\_log")))  
 test\_data <- eval(parse(text = paste0("test\_junction", j, "$Vehicles\_log")))  
   
 # Fit the ARIMA model  
 result\_junction\_arima <- fit\_arima\_model(train\_data, test\_data)  
   
 # Save the model, predictions, and metrics in the lists  
 junction\_models\_arima[[as.character(j)]] <- result\_junction\_arima$model  
 junction\_predictions\_arima[[as.character(j)]] <- result\_junction\_arima$predictions  
 junction\_metrics\_arima[[as.character(j)]] <- result\_junction\_arima$metrics  
   
 # Add junction name to metrics  
 metrics <- cbind(Junction = as.character(j), result\_junction\_arima$metrics)  
   
 # Accumulate metrics in the total\_metrics\_arima dataframe  
 total\_metrics\_arima <- rbind(total\_metrics\_arima, data.frame(metrics))  
}  
  
# Print the metrics for each junction with ARIMA  
for (j in 1:3) {  
 cat("\nMetrics for Junction", j, "with ARIMA:\n")  
 print(junction\_metrics\_arima[[as.character(j)]])  
}

##   
## Metrics for Junction 1 with ARIMA:  
## MAE MSE RMSE   
## 0.3126863 0.1345307 0.3667842   
##   
## Metrics for Junction 2 with ARIMA:  
## MAE MSE RMSE   
## 0.2921633 0.1192132 0.3452727   
##   
## Metrics for Junction 3 with ARIMA:  
## MAE MSE RMSE   
## 0.4216769 0.2687715 0.5184318

# Print the final dataframe with metrics for all junctions with ARIMA  
print("Total Metrics ARIMA:")

## [1] "Total Metrics ARIMA:"

print(total\_metrics\_arima)

## Junction V2  
## MAE 1 0.31268631414701  
## MSE 1 0.134530658087061  
## RMSE 1 0.366784211883583  
## MAE1 2 0.292163331774594  
## MSE1 2 0.119213212546637  
## RMSE1 2 0.345272664059345  
## MAE2 3 0.421676923083628  
## MSE2 3 0.268771513798262  
## RMSE2 3 0.518431783167527

# SARIMA Model  
  
# Initialize lists to store models, predictions, and metrics for each junction  
junction\_models\_sarima <- list()  
junction\_predictions\_sarima <- list()  
junction\_metrics\_sarima <- list()  
  
# Define a function for fitting and forecasting SARIMA  
fit\_sarima\_model <- function(train\_data, test\_data) {  
 # Fit a SARIMA model  
 sarima\_model <- Arima(train\_data, order = c(1, 1, 1), seasonal = list(order = c(1, 1, 1), period = 12))  
   
 # Make predictions on the test set  
 predictions\_sarima <- forecast(sarima\_model, h = length(test\_data))  
   
 # Calculate metrics  
 metrics <- calculate\_metrics(test\_data, predictions\_sarima$mean)  
   
 # Return the model, predictions, and metrics  
 return(list(model = sarima\_model, predictions = predictions\_sarima, metrics = metrics))  
}  
  
# Initialize an empty dataframe to store metrics  
total\_metrics\_sarima <- data.frame(Junction = character(0), MAE = numeric(0), MSE = numeric(0), RMSE = numeric(0))  
  
# Iterate over each junction  
for (j in 1:3) {  
 # Get the train and test data for the current junction  
 train\_data <- eval(parse(text = paste0("train\_junction", j, "$Vehicles\_log")))  
 test\_data <- eval(parse(text = paste0("test\_junction", j, "$Vehicles\_log")))  
   
 # Fit the SARIMA model  
 result\_junction\_sarima <- fit\_sarima\_model(train\_data, test\_data)  
   
 # Save the model, predictions, and metrics in the lists  
 junction\_models\_sarima[[as.character(j)]] <- result\_junction\_sarima$model  
 junction\_predictions\_sarima[[as.character(j)]] <- result\_junction\_sarima$predictions  
 junction\_metrics\_sarima[[as.character(j)]] <- result\_junction\_sarima$metrics  
   
 # Add junction name to metrics  
 metrics <- cbind(Junction = as.character(j), result\_junction\_sarima$metrics)  
   
 # Accumulate metrics in the total\_metrics\_sarima dataframe  
 total\_metrics\_sarima <- rbind(total\_metrics\_sarima, data.frame(metrics))  
}  
  
# Print the metrics for each junction with SARIMA  
for (j in 1:3) {  
 cat("\nMetrics for Junction", j, "with SARIMA:\n")  
 print(junction\_metrics\_sarima[[as.character(j)]])  
}

##   
## Metrics for Junction 1 with SARIMA:  
## MAE MSE RMSE   
## 0.3159922 0.1408275 0.3752699   
##   
## Metrics for Junction 2 with SARIMA:  
## MAE MSE RMSE   
## 0.5405169 0.4326395 0.6577534   
##   
## Metrics for Junction 3 with SARIMA:  
## MAE MSE RMSE   
## 0.4228157 0.2936567 0.5419010

# Print the final dataframe with metrics for all junctions with SARIMA  
print("Total Metrics SARIMA:")

## [1] "Total Metrics SARIMA:"

print(total\_metrics\_sarima)

## Junction V2  
## MAE 1 0.315992171600113  
## MSE 1 0.140827511167906  
## RMSE 1 0.375269917749752  
## MAE1 2 0.540516942958778  
## MSE1 2 0.432639538610448  
## RMSE1 2 0.65775340258371  
## MAE2 3 0.422815680334424  
## MSE2 3 0.293656663735838  
## RMSE2 3 0.541900972259543

#Evaluation Metrics to Seasonal Naive  
  
# Function to calculate MAE, MSE, and RMSE  
calculate\_metrics <- function(actual, forecast) {  
 errors <- actual - forecast  
 mae <- mean(abs(errors), na.rm = TRUE)  
 mse <- mean(errors^2, na.rm = TRUE)  
 rmse <- sqrt(mse)  
   
 return(c(MAE = mae, MSE = mse, RMSE = rmse))  
}  
  
# Seasonal Naive Model  
  
# Set the percentage of data for training (e.g., 80%)  
train\_percentage <- 0.8  
  
# Initialize a list to store models and predictions for each junction  
junction\_models\_sn <- list()  
junction\_predictions\_sn <- list()  
junction\_metrics <- list()  
  
# Define a function for fitting and forecasting  
fit\_sn\_model <- function(train\_data, test\_data) {  
 # Fit a Seasonal Naive model  
 sn\_model <- snaive(train\_data, h = length(test\_data))  
   
 # Make predictions on the test set  
 predictions\_sn <- forecast(sn\_model)  
   
 # Calculate metrics  
 metrics <- calculate\_metrics(test\_data, predictions\_sn$mean)  
   
 # Return the model, predictions, and metrics  
 return(list(model = sn\_model, predictions = predictions\_sn, metrics = metrics))  
}  
  
# Iterate over each junction  
for (j in 1:3) {  
 # Get the train and test data for the current junction  
 train\_data <- eval(parse(text = paste0("train\_junction", j, "$Vehicles\_log")))  
 test\_data <- eval(parse(text = paste0("test\_junction", j, "$Vehicles\_log")))  
   
 # Fit the Seasonal Naive model  
 result\_junction <- fit\_sn\_model(train\_data, test\_data)  
   
 # Save the model, predictions, and metrics in the lists  
 junction\_models\_sn[[as.character(j)]] <- result\_junction$model  
 junction\_predictions\_sn[[as.character(j)]] <- result\_junction$predictions  
 junction\_metrics[[as.character(j)]] <- result\_junction$metrics  
}  
  
# Print the metrics for each junction  
for (j in 1:3) {  
 cat("\nMetrics for Junction", j, ":\n")  
 print(junction\_metrics[[as.character(j)]])  
}

##   
## Metrics for Junction 1 :  
## MAE MSE RMSE   
## 0.3128057 0.1339939 0.3660518   
##   
## Metrics for Junction 2 :  
## MAE MSE RMSE   
## 0.3079644 0.1405705 0.3749273   
##   
## Metrics for Junction 3 :  
## MAE MSE RMSE   
## 0.4236349 0.2792485 0.5284397

# Evaluation Metrics to Seasonal Naive  
  
# Function to calculate MAE, MSE, and RMSE  
calculate\_metrics <- function(actual, forecast) {  
 errors <- actual - forecast  
 mae <- mean(abs(errors), na.rm = TRUE)  
 mse <- mean(errors^2, na.rm = TRUE)  
 rmse <- sqrt(mse)  
   
 return(c(MAE = mae, MSE = mse, RMSE = rmse))  
}  
  
# Seasonal Naive Model  
  
# Set the percentage of data for training (e.g., 80%)  
train\_percentage <- 0.8  
  
# Initialize lists to store models, predictions, and metrics for each junction  
junction\_models\_sn <- list()  
junction\_predictions\_sn <- list()  
junction\_metrics <- list()  
  
# Define a function for fitting and forecasting  
fit\_sn\_model <- function(train\_data, test\_data) {  
 # Fit a Seasonal Naive model  
 sn\_model <- snaive(train\_data, h = length(test\_data))  
   
 # Make predictions on the test set  
 predictions\_sn <- forecast(sn\_model)  
   
 # Calculate metrics  
 metrics <- calculate\_metrics(test\_data, predictions\_sn$mean)  
   
 # Return the model, predictions, and metrics  
 return(list(model = sn\_model, predictions = predictions\_sn, metrics = metrics))  
}  
  
# Initialize an empty dataframe to store metrics  
total\_metrics\_sn <- data.frame(Junction = character(0), MAE = numeric(0), MSE = numeric(0), RMSE = numeric(0))  
  
# Iterate over each junction  
for (j in 1:3) {  
 # Get the train and test data for the current junction  
 train\_data <- eval(parse(text = paste0("train\_junction", j, "$Vehicles\_log")))  
 test\_data <- eval(parse(text = paste0("test\_junction", j, "$Vehicles\_log")))  
   
 # Fit the Seasonal Naive model  
 result\_junction <- fit\_sn\_model(train\_data, test\_data)  
   
 # Save the model, predictions, and metrics in the lists  
 junction\_models\_sn[[as.character(j)]] <- result\_junction$model  
 junction\_predictions\_sn[[as.character(j)]] <- result\_junction$predictions  
 junction\_metrics[[as.character(j)]] <- result\_junction$metrics  
   
 # Add junction name to metrics  
 metrics <- cbind(Junction = as.character(j), result\_junction$metrics)  
   
 # Accumulate metrics in the total\_metrics\_sn dataframe  
 total\_metrics\_sn <- rbind(total\_metrics\_sn, data.frame(metrics))  
}  
  
# Print the metrics for each junction  
for (j in 1:3) {  
 cat("\nMetrics for Junction", j, ":\n")  
 print(junction\_metrics[[as.character(j)]])  
}

##   
## Metrics for Junction 1 :  
## MAE MSE RMSE   
## 0.3128057 0.1339939 0.3660518   
##   
## Metrics for Junction 2 :  
## MAE MSE RMSE   
## 0.3079644 0.1405705 0.3749273   
##   
## Metrics for Junction 3 :  
## MAE MSE RMSE   
## 0.4236349 0.2792485 0.5284397

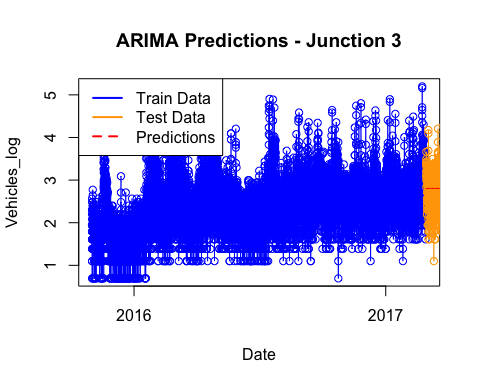
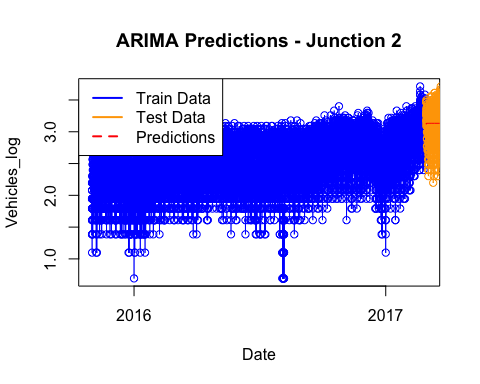
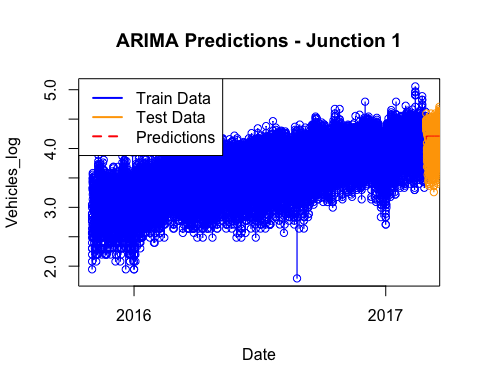
# Print the final dataframe with metrics for all junctions  
print("Total Metrics SN:")

## [1] "Total Metrics SN:"

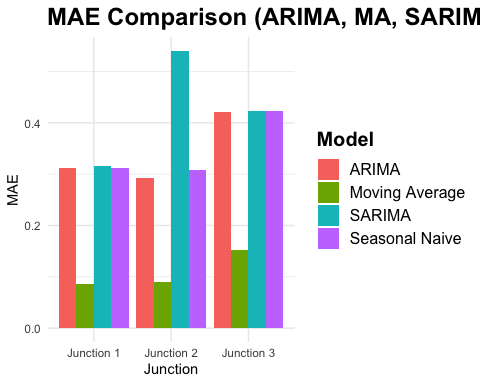
print(total\_metrics\_sn)

## Junction V2  
## MAE 1 0.312805682935709  
## MSE 1 0.133993898881455  
## RMSE 1 0.366051770766724  
## MAE1 2 0.307964435338376  
## MSE1 2 0.140570508626711  
## RMSE1 2 0.374927337795887  
## MAE2 3 0.423634917343435  
## MSE2 3 0.279248544429734  
## RMSE2 3 0.528439726392456

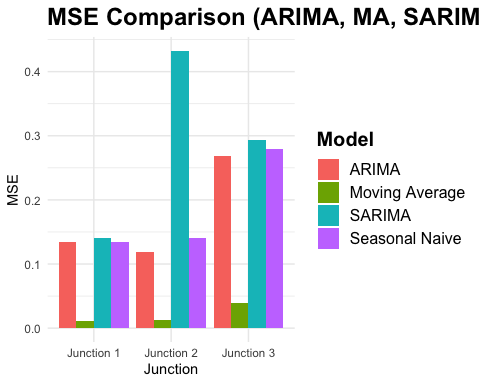
#plotting ARIMA predictions  
plot\_arima\_predictions <- function(train\_data, test\_data, predictions, junction\_num, line\_width = 0.75, plot\_width = 10, plot\_height = 5) {  
 # Convert DateTime to a Date class for plotting  
 train\_data$DateTime <- as.Date(train\_data$DateTime)  
 test\_data$DateTime <- as.Date(test\_data$DateTime)  
   
 # Set plot size  
 options(repr.plot.width = plot\_width, repr.plot.height = plot\_height)  
   
 # Plot the time series, training data, and predictions  
 plot(train\_data$DateTime, train\_data$Vehicles\_log, type = "o", col = "blue", lwd = line\_width,  
 xlab = "Date", ylab = "Vehicles\_log", main = paste("ARIMA Predictions - Junction", junction\_num))  
   
 # Add test data to the plot  
 points(test\_data$DateTime, test\_data$Vehicles\_log, type = "o", col = "orange", lwd = line\_width)  
   
 # Add predictions to the plot  
 lines(test\_data$DateTime, predictions$mean, col = "red", lty = 5, lwd = line\_width)  
   
# Add legend with smaller text size  
legend("topleft", legend = c("Train Data", "Test Data", "Predictions"),   
 col = c("blue", "orange", "red"), lty = c(1, 1, 2), lwd = line\_width \* 2, cex = 1)  
}  
  
# Iterate over each junction to plot ARIMA predictions  
for (j in 1:3) {  
 # Get the train, test data, and predictions for the current junction  
 train\_data <- eval(parse(text = paste0("train\_junction", j)))  
 test\_data <- eval(parse(text = paste0("test\_junction", j)))  
 predictions <- junction\_predictions\_arima[[as.character(j)]]  
   
 # Plot ARIMA predictions with adjusted parameters  
 plot\_arima\_predictions(train\_data, test\_data, predictions, junction\_num = j, line\_width = 1, plot\_width = 20, plot\_height = 7)  
}



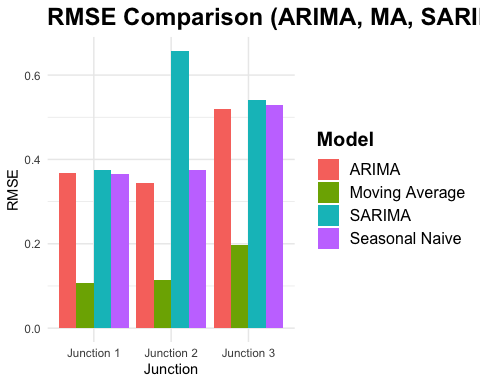
# Data for Moving Average model  
total\_errors$Metric <- c("MAE", "MSE", "RMSE", "MAE", "MSE", "RMSE", "MAE", "MSE", "RMSE")  
mae\_ma <- as.numeric(total\_errors$errors[total\_errors$Metric == 'MAE'])  
mse\_ma <- as.numeric(total\_errors$errors[total\_errors$Metric == 'MSE'])  
rmse\_ma <- as.numeric(total\_errors$errors[total\_errors$Metric == 'RMSE'])  
  
# Data for ARIMA model  
total\_metrics\_arima$Metric <- c("MAE", "MSE", "RMSE", "MAE", "MSE", "RMSE", "MAE", "MSE", "RMSE")  
mae\_arima <- as.numeric(total\_metrics\_arima$V2[total\_metrics\_arima$Metric == 'MAE'])  
mse\_arima <- as.numeric(total\_metrics\_arima$V2[total\_metrics\_arima$Metric == 'MSE'])  
rmse\_arima <- as.numeric(total\_metrics\_arima$V2[total\_metrics\_arima$Metric == 'RMSE'])  
  
# Data for SARIMA model  
total\_metrics\_sarima$Metric <- c("MAE", "MSE", "RMSE", "MAE", "MSE", "RMSE", "MAE", "MSE", "RMSE")  
mae\_sarima <- as.numeric(total\_metrics\_sarima$V2[total\_metrics\_sarima$Metric == 'MAE'])  
mse\_sarima <- as.numeric(total\_metrics\_sarima$V2[total\_metrics\_sarima$Metric == 'MSE'])  
rmse\_sarima <- as.numeric(total\_metrics\_sarima$V2[total\_metrics\_sarima$Metric == 'RMSE'])  
  
# Data for Seasonal Naive model  
total\_metrics\_sn$Metric <- c("MAE", "MSE", "RMSE", "MAE", "MSE", "RMSE", "MAE", "MSE", "RMSE")  
mae\_seasonal\_naive <- as.numeric(total\_metrics\_sn$V2[total\_metrics\_sn$Metric == 'MAE'])  
mse\_seasonal\_naive <- as.numeric(total\_metrics\_sn$V2[total\_metrics\_sn$Metric == 'MSE'])  
rmse\_seasonal\_naive <- as.numeric(total\_metrics\_sn$V2[total\_metrics\_sn$Metric == 'RMSE'])  
  
# Create a data frame for ARIMA model  
data\_arima <- data.frame(Junction = c("Junction 1", "Junction 2", "Junction 3"),  
 Model = rep("ARIMA", 3),  
 MAE = mae\_arima, MSE = mse\_arima, RMSE = rmse\_arima)  
  
# Create a data frame for SARIMA model  
data\_sarima <- data.frame(Junction = c("Junction 1", "Junction 2", "Junction 3"),  
 Model = rep("SARIMA", 3),  
 MAE = mae\_sarima, MSE = mse\_sarima, RMSE = rmse\_sarima)  
  
# Create a data frame for Seasonal Naive model  
data\_seasonal\_naive <- data.frame(Junction = c("Junction 1", "Junction 2", "Junction 3"),  
 Model = rep("Seasonal Naive", 3),  
 MAE = mae\_seasonal\_naive, MSE = mse\_seasonal\_naive, RMSE = rmse\_seasonal\_naive)  
  
# Create a data frame for Moving Average model  
data\_ma <- data.frame(Junction = c("Junction 1", "Junction 2", "Junction 3"),  
 Model = rep("Moving Average", 3),  
 MAE = mae\_ma, MSE = mse\_ma, RMSE = rmse\_ma)  
  
# Combine all data frames including Moving Average model  
data\_combined\_all <- rbind(data\_arima, data\_sarima, data\_seasonal\_naive, data\_ma)  
  
# Function to create a grouped bar chart for all models with increased font size  
create\_grouped\_bar\_chart\_all <- function(metric, title) {  
 ggplot(data\_combined\_all, aes(x = Junction, y = !!sym(metric), fill = Model)) +  
 geom\_bar(stat = "identity", position = "dodge") +  
 labs(title = title, y = metric) +  
 theme\_minimal() +  
 theme(legend.text = element\_text(size = 12),  
 legend.title = element\_text(size = 15, face = "bold"),  
 plot.title = element\_text(size = 18, face = "bold"))  
}  
  
# Print the faceted bar chart for all metrics with increased font size  
print(create\_grouped\_bar\_chart\_all("MAE", "MAE Comparison (ARIMA, MA, SARIMA, Seasonal Naive)"))



print(create\_grouped\_bar\_chart\_all("MSE", "MSE Comparison (ARIMA, MA, SARIMA, Seasonal Naive)"))



print(create\_grouped\_bar\_chart\_all("RMSE", "RMSE Comparison (ARIMA, MA, SARIMA, Seasonal Naive)"))



# Print the summary table  
summary\_table <- data\_combined\_all %>% arrange(data\_combined\_all$Junction)  
print(summary\_table)

## Junction Model MAE MSE RMSE  
## 1 Junction 1 ARIMA 0.31268631 0.13453066 0.3667842  
## 2 Junction 1 SARIMA 0.31599217 0.14082751 0.3752699  
## 3 Junction 1 Seasonal Naive 0.31280568 0.13399390 0.3660518  
## 4 Junction 1 Moving Average 0.08596167 0.01158036 0.1076121  
## 5 Junction 2 ARIMA 0.29216333 0.11921321 0.3452727  
## 6 Junction 2 SARIMA 0.54051694 0.43263954 0.6577534  
## 7 Junction 2 Seasonal Naive 0.30796444 0.14057051 0.3749273  
## 8 Junction 2 Moving Average 0.08965130 0.01285185 0.1133660  
## 9 Junction 3 ARIMA 0.42167692 0.26877151 0.5184318  
## 10 Junction 3 SARIMA 0.42281568 0.29365666 0.5419010  
## 11 Junction 3 Seasonal Naive 0.42363492 0.27924854 0.5284397  
## 12 Junction 3 Moving Average 0.15229755 0.03881555 0.1970166

# Print the table  
kable(data\_combined\_all, "markdown")

| Junction | Model | MAE | MSE | RMSE |
| --- | --- | --- | --- | --- |
| Junction 1 | ARIMA | 0.3126863 | 0.1345307 | 0.3667842 |
| Junction 2 | ARIMA | 0.2921633 | 0.1192132 | 0.3452727 |
| Junction 3 | ARIMA | 0.4216769 | 0.2687715 | 0.5184318 |
| Junction 1 | SARIMA | 0.3159922 | 0.1408275 | 0.3752699 |
| Junction 2 | SARIMA | 0.5405169 | 0.4326395 | 0.6577534 |
| Junction 3 | SARIMA | 0.4228157 | 0.2936567 | 0.5419010 |
| Junction 1 | Seasonal Naive | 0.3128057 | 0.1339939 | 0.3660518 |
| Junction 2 | Seasonal Naive | 0.3079644 | 0.1405705 | 0.3749273 |
| Junction 3 | Seasonal Naive | 0.4236349 | 0.2792485 | 0.5284397 |
| Junction 1 | Moving Average | 0.0859617 | 0.0115804 | 0.1076121 |
| Junction 2 | Moving Average | 0.0896513 | 0.0128518 | 0.1133660 |
| Junction 3 | Moving Average | 0.1522976 | 0.0388155 | 0.1970166 |