RIDING AHEAD OF DEMAND: A SMART FORECASTING APPROACH TO DIVVY BIKE SHORTAGES

A PROJECT REPORT

Submitted by

SANJANA WAGHRAY - A20576599

SHARANYA MISHRA- A20579993

HARISH NAMASIVAYAM MUTHUSWAMY - A20588339

ARUNESHWARAN SIVAKUMAR - A20593046

BHOOPLAM PRAVEEN GAYATHRI ANANYA - A20588605

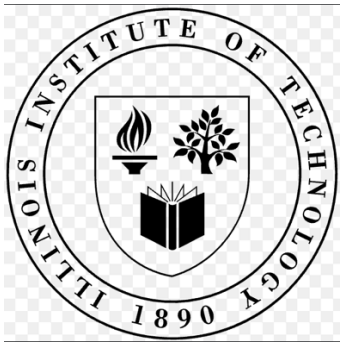
For the course

MATH 546 - INTRODUCTION TO TIME SERIES

During the

SPRING 2025 SEMESTER

At



2.1. Preliminary Discussion and Analysis

2.1.1. Description of the Dataset

Our dataset comes from Divvy, bike-sharing system in the United States. It contains details about every ride taken between **January 1st to June 30th, 2024**. The raw data is originally collected month-wise and includes columns like ride ID, start time, end time, user type, and station names. We combined these monthly files and filtered only the data for **member users**, as this group showed more consistent patterns.

After combining and filtering, we created a daily ride count column by grouping rides based on their start date. This gave us one row for each date with the total number of rides taken that day.

When we plotted the data, two features stood out:

1. **Upward trend**: Ride counts increased steadily from January to June. This might be due to improving weather, more people subscribing, or increased promotions.
2. **Seasonality**: There was a clear 7-day repeating pattern, which makes sense because people use bikes differently on weekdays compared to weekends.

There were also a few sudden dips and spikes. These could be due to extreme weather, public holidays, or technical issues in the system.

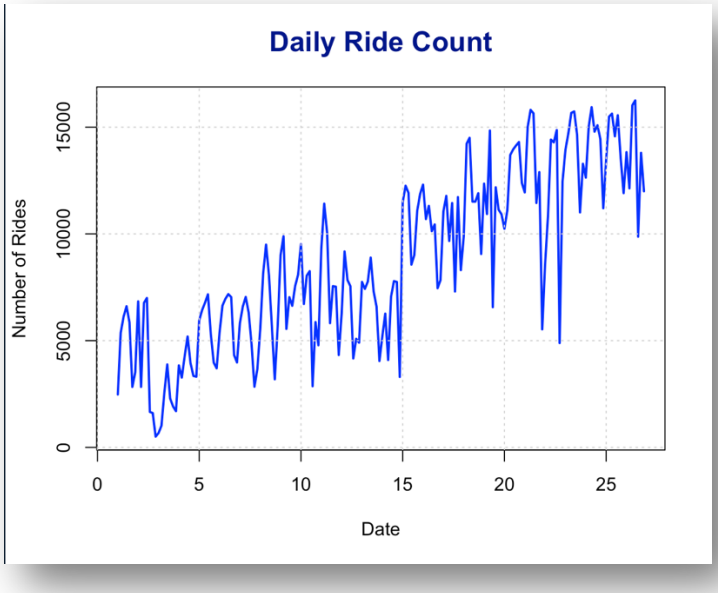


Figure 1: Line plot of daily ride counts (Raw Data)

2.1.2. Preliminary Transformations and Differencing Order

Time series models usually work best when the data is **stationary**. That means the average and variance of the data should stay roughly the same over time. But our data clearly wasn't stationary:

- The average number of rides increased as we moved from January to June.
- The ride counts had more variation in later months, which shows increasing variance.

To fix the changing variance, we applied a **Box-Cox transformation**. This transformation helps stabilize the scale of the data. The BoxCox.lambda() function suggested a lambda value of **0.7**, so we used that.

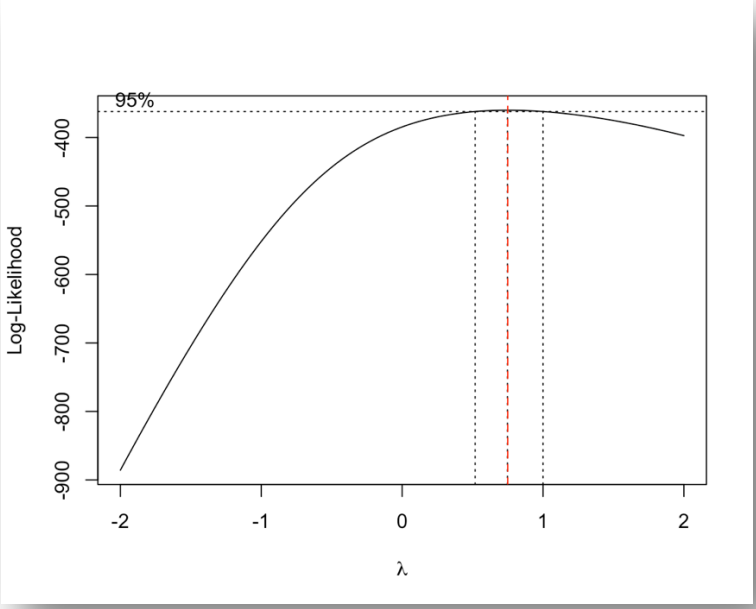


Figure 2: Box-Cox lambda estimation plot

After that, we took care of the **weekly seasonality** by applying **seasonal differencing with a lag of 7**. This means we subtracted the value from 7 days earlier from the current day's value. This helped remove the repeating weekly cycle.

We ran the **Augmented Dickey-Fuller (ADF) test** to check if the transformed and differenced data was now stationary. The p-value from the test was low, which means we can assume that the data doesn't require further differencing.

```
> adfTest(seasonal_diff, type = "c")
Warning in adfTest(seasonal_diff, type = "c") :
  p-value smaller than printed p-value

Title:
 Augmented Dickey-Fuller Test

Test Results:
  PARAMETER:
    Lag Order: 1
  STATISTIC:
    Dickey-Fuller: -6.6731
  P VALUE:
    0.01

Description:
 Fri Apr 25 20:49:44 2025 by user:
```

\

Figure 3: ADF test result / summary

2.1.3. Sample ACF and PACF Plots

Once we had a stationary series, we looked at its **ACF (AutoCorrelation Function)** and **PACF (Partial AutoCorrelation Function)** plots. These plots help us decide what model structure might work best.

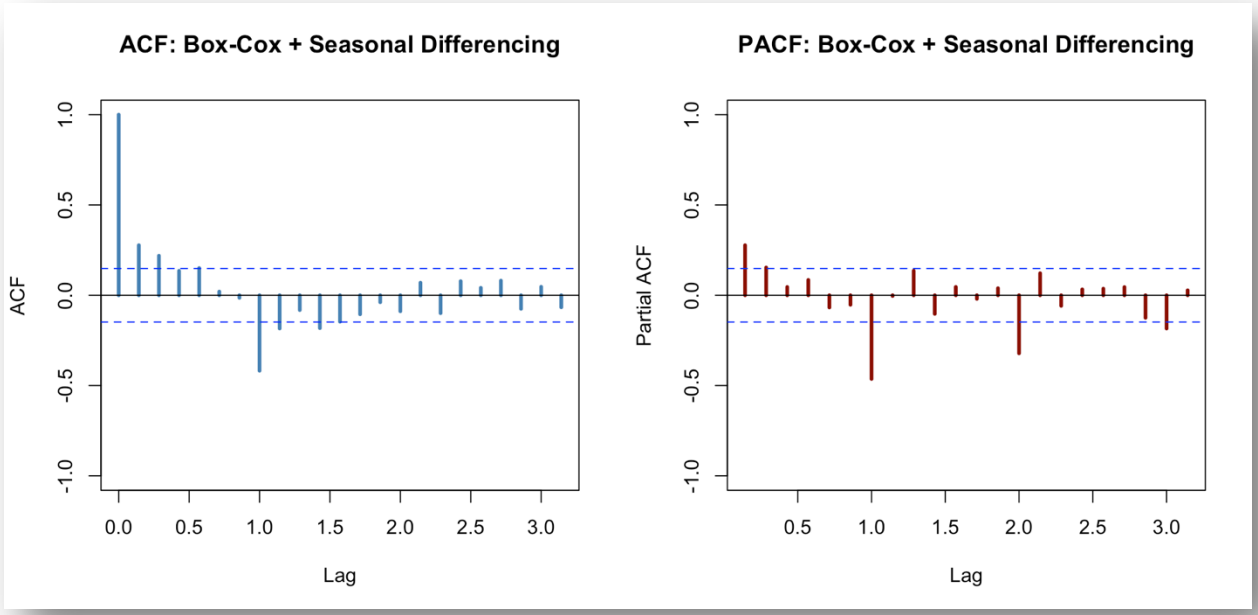


Figure 4: ACF & PACF of transformed and differenced series

Here’s what we noticed:

- The **ACF plot** still shows a **clear drop-off after a strong initial spike at lag 1**, and while the **weekly seasonal pattern (lag 7)** is not as strongly visible as before, there's **still mild decay**, which can suggest **short-term persistence**.
- The **PACF plot** cuts off sharply after **lag 1**, with remaining lags being close to zero or within the confidence bounds. This again hints at a **possible AR(1)** or **ARIMA(p, d, 0)**-type structure.

These plots help in identifying the appropriate model order. The sharp PACF cut-off at lag 1 supports the inclusion of an AR term, and the absence of significant spikes in ACF beyond lag 1 suggests minimal MA structure after seasonal differencing. This suggests the series is now well-prepared for ARIMA modeling.

2.2. Fitting the Model

2.2.1 Model 1: ARIMA (2,0,0)

$$(1 - 0.2151B - 0.1574B^2)X_t = W_t$$

; where **Wt € WN(0, 6926869)**

We started by trying a regular **ARIMA (2,0,0)** model. This model type uses 2 autoregressive terms, 0 differencing, and 0 moving average terms. It seemed like a good structure for capturing short-term changes.

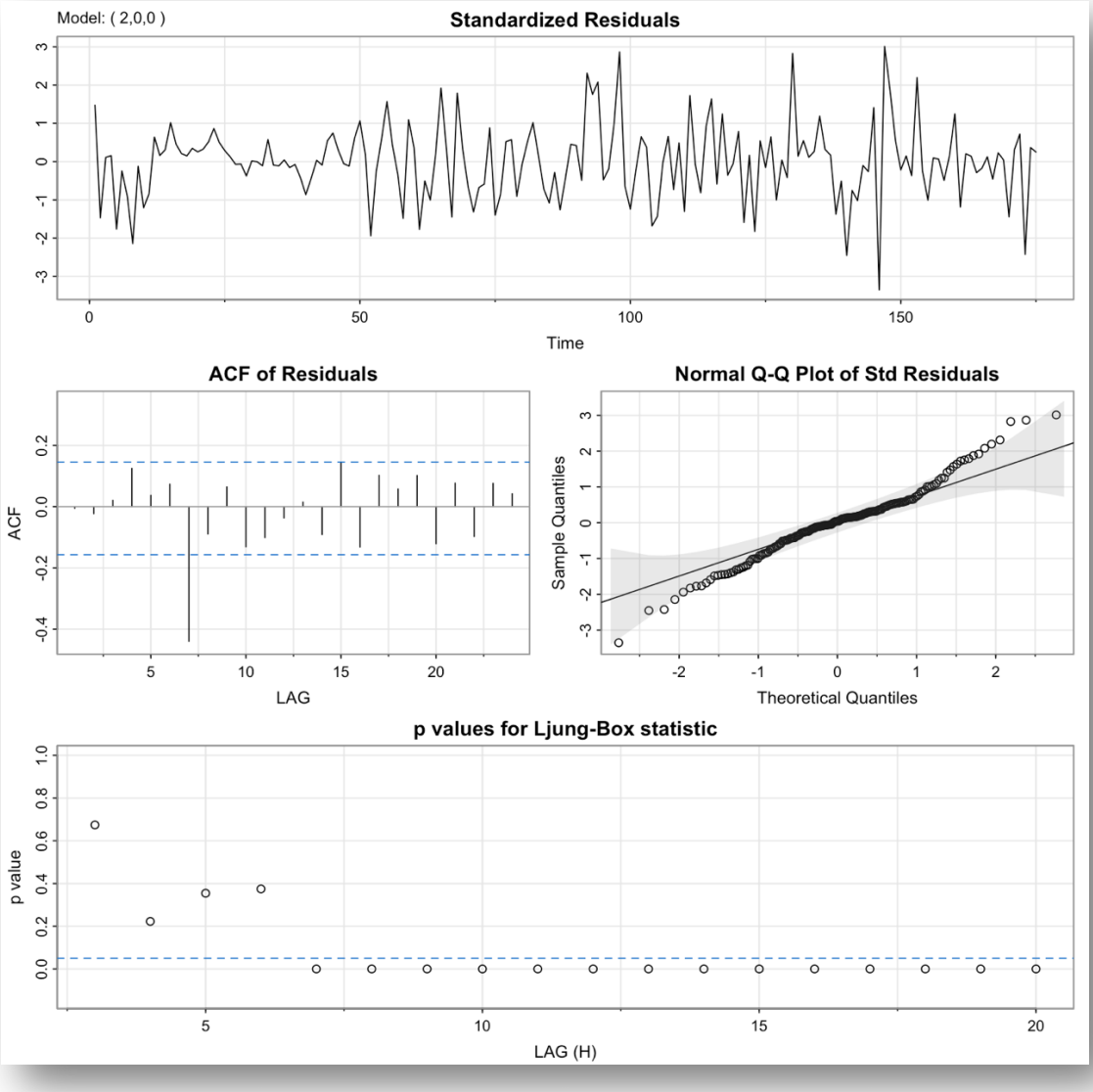


Figure 5: SARIMA plot for model 1

As visible from the above plot- we rejected this because of the following reasons:

- The **Ljung–Box test** showed bad results.
- The **ACF of residuals** do not follow white noise as few of the lines are outside the interval.

Model Summary:

```
ARIMA(2,0,0) with zero mean

Coefficients:
         ar1     ar2
      0.2151  0.1574
s.e.  0.0751  0.0752

sigma^2 = 6926869:  log likelihood = -1625.57
AIC=3257.15   AICc=3257.29   BIC=3266.64

Training set error measures:
                    ME     RMSE      MAE      MPE     MAPE      MASE        ACF1
Training set 217.6398 2616.812 1923.786 76.89944 123.8247 0.7796098 -0.01343023
```

Figure 6: Model 1 summary

Visibly, it didn't perform very well. The residuals (the part of the data the model couldn't explain) still showed some patterns. This means the model missed some structure, possibly the weekly seasonality. This showed us that a **seasonal model** might do better. Hence, we finalized to move forward with SARIMA models and set frequency = 7 to capture the weekly pattern.

2.2.2 Trying Seasonal Models (SARIMA)

Using the auto.arima() function with seasonal = TRUE, we tried several seasonal models. Two of them stood out:

- SARIMA(2,0,0)(2,0,2)[7]
- SARIMA(1,0,1)(2,0,2)[7]

These models include seasonal terms to account for the weekly pattern we observed earlier. We applied these models to the transformed and seasonally differenced data (seasonal_diff).

- **Model 2: SARIMA (2,0,0) (2, 0, 2) [7]**

$$(1 - 0.3449B)(1 - 0.822B^7 + 0.0486B^{14})X_t = (1 - 1.6265B^7 + 0.7309B^{14})W_t$$ **; where**

Wt € WN(0, 49911)

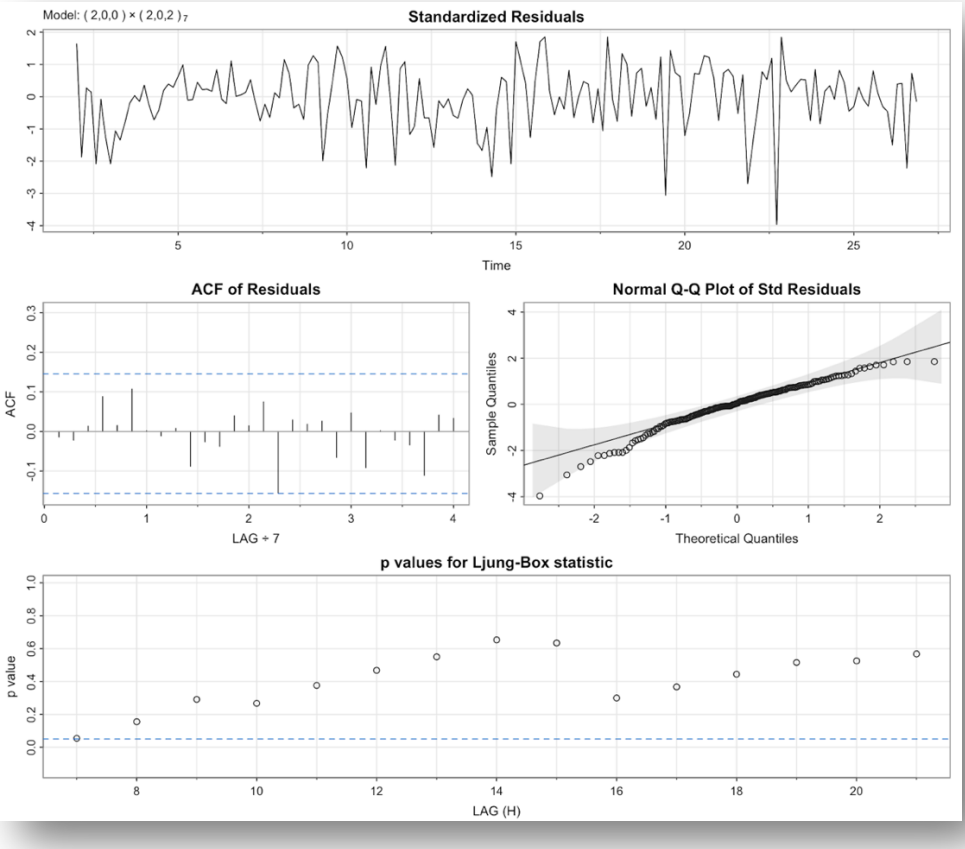


Figure 7: SARIMA plot for model 2

From the above plot we conclude that the diagnostic plots for this model look good and indicate a strong fit.

Model Summary:

```
Series: seasonal_diff
ARIMA(2,0,0)(2,0,2)[7] with zero mean

Coefficients:
         ar1     ar2   sar1     sar2     sma1    sma2
      0.3449  0.2468  0.822  -0.0486  -1.6265  0.7309
s.e.  0.0835  0.0809  0.217   0.1278   0.1979  0.1354

sigma^2 = 49911:  log likelihood = -1195.68
AIC=2405.36   AICc=2406.03   BIC=2427.52

Training set error measures:
                    ME     RMSE      MAE      MPE     MAPE      MASE        ACF1
Training set 38.47123 219.5451 170.0811 51.84711 164.2034 0.4543563 -0.0622164
>
```

Figure 8: Model 2 Summary

- **Model 3: SARIMA (1,0,1) (2,0,2) [7]**

Equation:

$$(1-0.9804B)(1+0.4087B^7+0.1027B^{14})X_t = (1-0.6985B)(1-0.5304B^7-0.2724B^{14})W_t$$

$$X_t \in W(0,\sigma^2) \quad with \quad \sigma^2 = 46553$$

Using the auto.arima() function it suggested to go ahead with the above model. The models include seasonal terms to account for the weekly pattern we observed earlier. We applied these models to the transformed and seasonally differenced data (seasonal_diff).

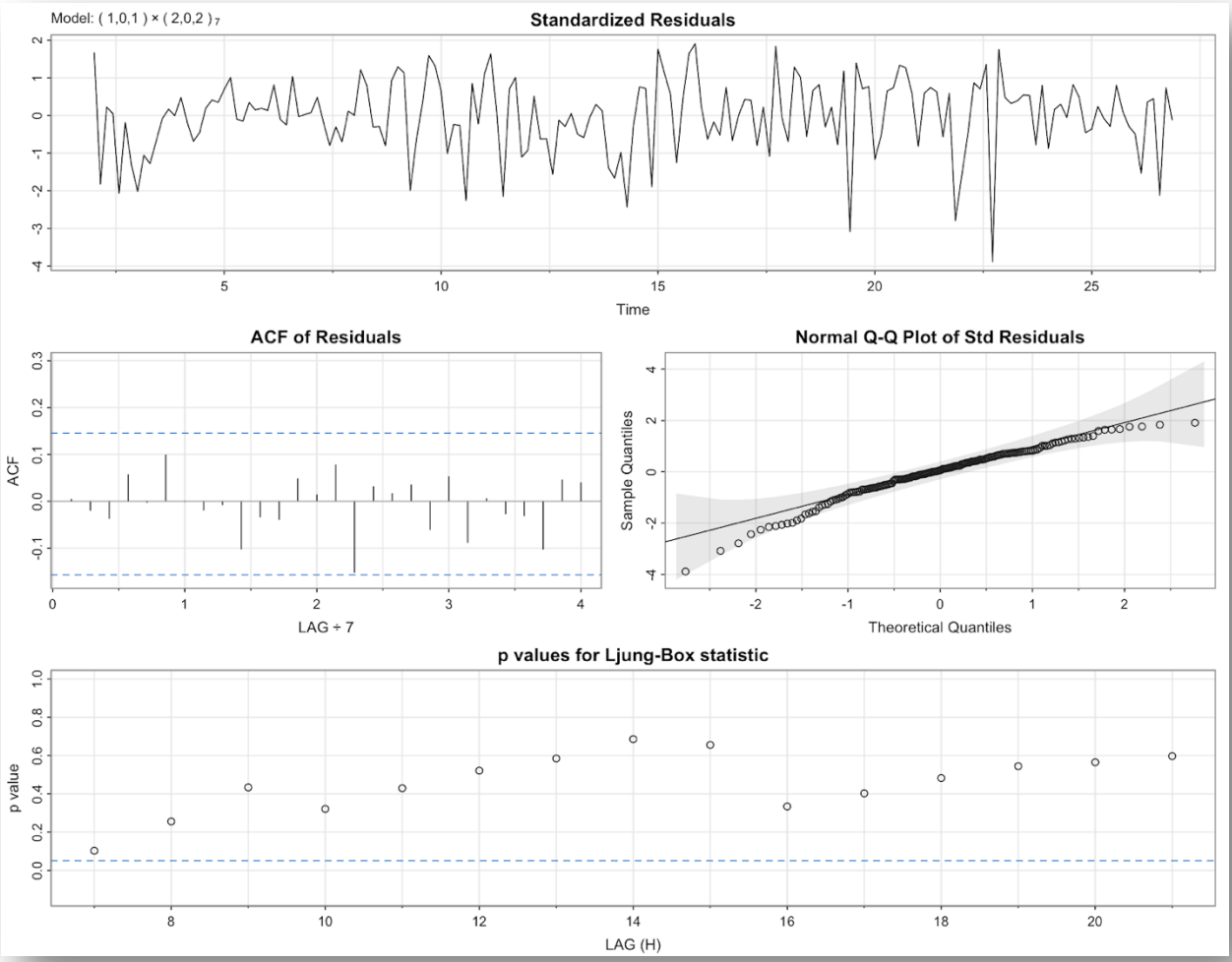


Figure 9: SARIMA plot for model 3

From the above graph we can conclude that-

- The **Ljung–Box test** showed no significant autocorrelation in residuals (all p-values > 0.05),
- The **ACF of residuals** confirmed white noise behavior,
- Q-Q plot of residuals suggested approximate normality.

Model Summary:

```
ARIMA(1,0,1)(2,0,2)[7] with zero mean

Coefficients:
         ar1      ma1     sar1     sar2     sma1     sma2
      0.9804  -0.6985  -0.4087  -0.1027  -0.5304  -0.2724
s.e.  0.0377   0.0856   0.6946   0.1193   0.6747   0.6764

sigma^2 = 46553:  log likelihood = -1190.51
AIC=2395.02   AICc=2395.69   BIC=2417.18

Training set error measures:
                    ME    RMSE      MAE      MPE     MAPE      MASE        ACF1
Training set 19.17896 212.03 166.2992 2.708351 183.0851 0.4442534 0.05160228
```

Figure 10: Model 3 summary

- **Final Model: SARIMA (1,0,1) (2,1,2) [7]**

After comparing the two, we finalized the model **SARIMA(1,0,1)(2,0,2)[7]** because it gave us a lower AIC and AICc than the other. This means it balanced the goodness-of-fit and model complexity better. The residuals from this model were clean and random, which is what we want. The output from auto.arima() also supported this structure. To make sure we properly handled the seasonality, we included **D = 1** in our final model, which means the model included seasonal differencing. All of this made SARIMA(1,0,1)(2,1,2)[7] a strong and justifiable choice for our final model.

Model Summary:

```
Series: ts_data
ARIMA(1,0,1)(2,1,2)[7]
Box Cox transformation: lambda= 0.7494908

Coefficients:
         ar1      ma1     sar1     sar2     sma1     sma2
      0.9804  -0.6985  -0.4087  -0.1027  -0.5304  -0.2724
s.e.  0.0377   0.0856   0.6946   0.1193   0.6747   0.6764

sigma^2 = 46553:  log likelihood = -1190.51
AIC=2395.02   AICc=2395.69   BIC=2417.18

Training set error measures:
                    ME     RMSE      MAE       MPE     MAPE      MASE       ACF1
Training set 211.475 1975.082 1514.087 -5.922079 25.03462 0.7272749 0.02646255
```

Figure 11: Final Model summary

2.3. Forecasting

Once the model was fitted, we used it to forecast the number of rides for the next **14 days (July 1 to July 14, 2024)** on our Pre-Transformed Series Dataset and got the following results.

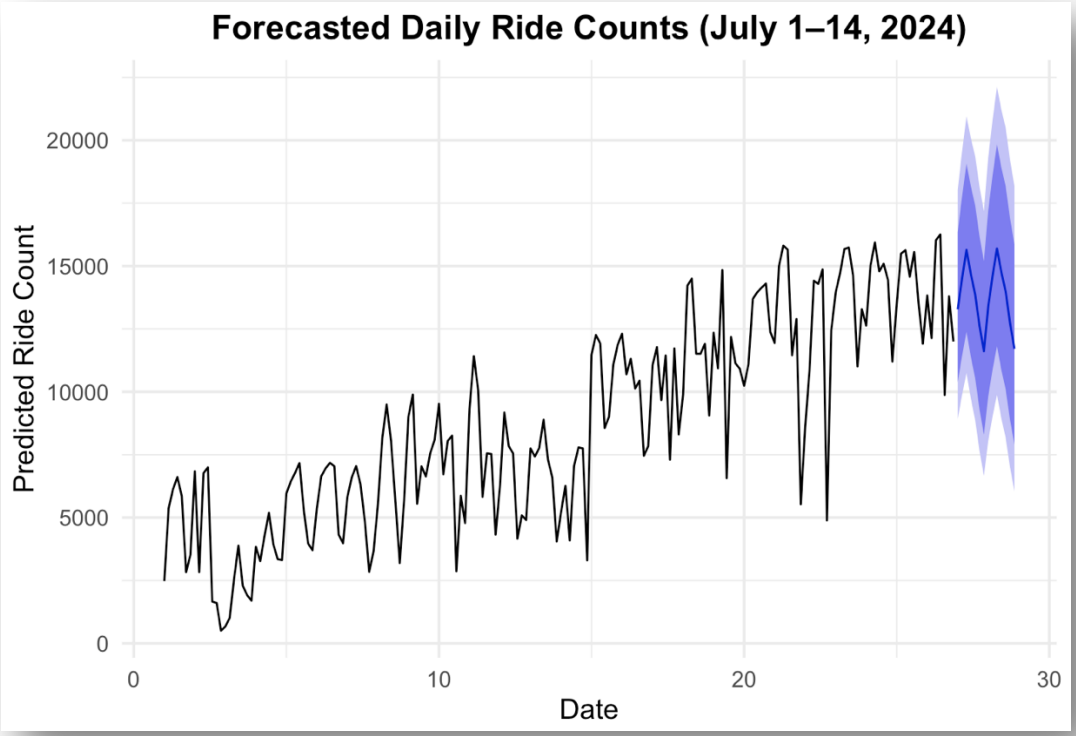


Figure 12: Forecast with Confidence Intervals

We then compared the forecasted values with the actual data.

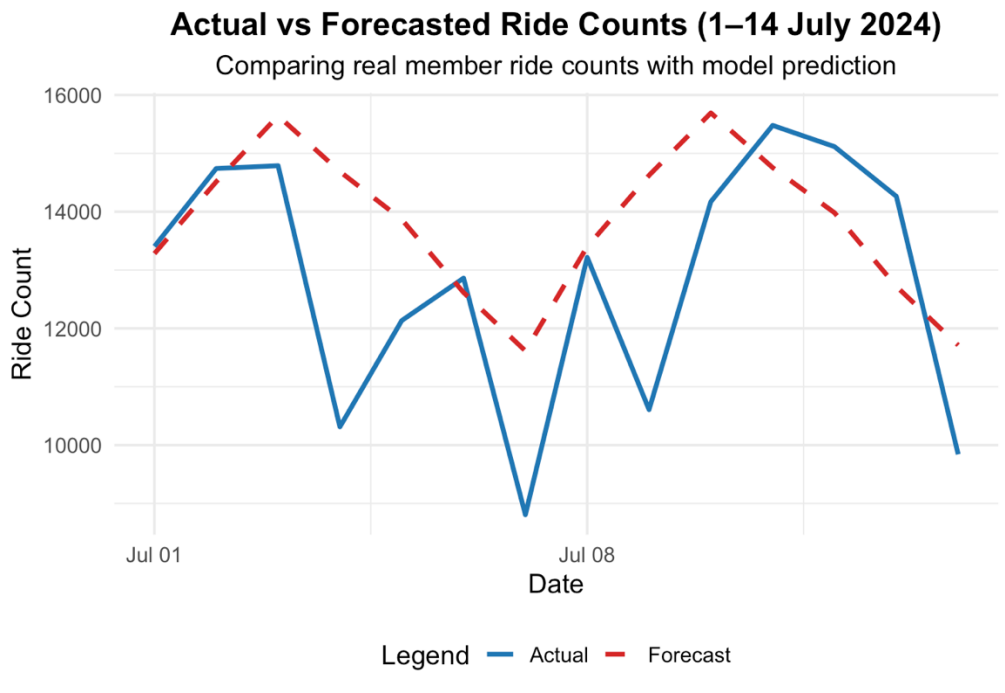


Figure 13: Actual vs Forecasted

The forecast followed the overall pattern of the actual values. It didn't match every spike and dip perfectly, but it stayed close to the trend, which can be seen in the comparison above. So, the forecast turned out to be pretty accurate.

The error metrics were:

```
> cat("Mean Absolute Error (MAE):", mae, "\n")
Mean Absolute Error (MAE): 1525.638
> cat("Root Mean Squared Error (RMSE):", rmse, "\n")
Root Mean Squared Error (RMSE): 2018.567
> cat("Mean Absolute Percentage Error (MAPE):", mape,"%")
Mean Absolute Percentage Error (MAPE): 13.6227 %
```

These are good results for short-term forecasting. The model can help Divvy predict demand and plan how to move bikes between stations.

Some interesting observations we made during the project, though not part of the diagnostic results:

- Setting frequency = 7 gave us a clearer picture of the weekly pattern in the data.
- Using type = "ct" in the ADF test consistently gave p-values below 0.05, likely because it accounts for both trend and intercept. This made the test always suggest no need for differencing even when it might be needed so it wasn't the best option for our case.
- Including too many data points at once added a lot of noise, which can hurt forecast accuracy.

2.3.1 Conclusion

In this project, we explored and forecasted Divvy bike ride data using time series modeling. We started by cleaning and transforming the data, applied Box-Cox and seasonal differencing, and checked for stationarity.

We tested different ARIMA and SARIMA models and finally selected **SARIMA (1,0,1) (2, 1, 2) [7]** as our best model. It passed all diagnostic checks and gave a reliable short-term forecast.

This model can help Divvy make practical decisions like predicting ride demand, keeping stations balanced, and planning services. We kept our approach straightforward, using basic transformations and testing different models to build a forecasting system that’s both reliable and easy to interpret.

2.4 Appendix

2.4.1 Appendix 1:

```
library(forecast) library(tseries) library(fUnitRoots) library(dplyr) library(lubridate)
library(astsa)
```

Step 1: Filter for Jan–June 2024

```
ts_data1 <- read.csv("./ts_data.csv")

#Figure 1 generated by
ts_data2 <- ts_data1[ts_data1$date >= as.Date("2024-01-01") & ts_data1$date <=
as.Date("2024-06-30"), ] ts_data <- ts(ts_data2$ride_count, frequency = 7) #View(ts_data)
```

Step 2: Directly utilising the dataset without introducing frequency:

```
plot(ts_data2$ride_count, type = "l", col = "blue", main = "Ride Count Per Day Janurary 2024-
June 2024", xlab = "Date", ylab = "Ride Count")

#a: checking if differencing required:

adfTest(ts_data2$ride_count, type = "nc")

#b: differencing:

ts_data2_difference<-diff(ts_data2$ride_count,lag=7)

#c: fitting the model
model_original <- auto.arima(ts_data2_difference,trace=TRUE)


#Figure 6 generated by
summary(model_original)

#Figure 5 generated by
sarima(ts_data2_difference, p = 2, d = 0, q = 0)
```

Step 3: Plotting the raw dataset with Frequency

```
plot(ts_data, type = "l", col = "blue", main = "Ride Count Per Day January 2024 - June 2024",
xlab = "Date", ylab = "Ride Count")
```

Step 4: Ensure ride_count has only positive values (Box-Cox doesn't allow zeros or negatives)

boxcox_result <- BoxCox.lambda(ts_data) print(paste("Optimal lambda:", boxcox_result)) lambda <- BoxCox.lambda(ts_data) ts_data$transformed <- BoxCox(ts_data, lambda)

library(MASS)

#Figure 2 generated by

boxcox(ts_data ~ 1, lambda = seq(-2, 2, 0.1), main = "Figure 2: Box-Cox Lambda Estimation", xlab = expression(lambda), ylab = "Log-Likelihood", lwd = 2, col = "blue") # better color

abline(v = BoxCox.lambda(ts_data), col = "red", lty = 2) # show optimal lambda

#Step 5: Differencing the Box-cox transformed Dataset

seasonal_diff <- diff(ts_data$transformed, lag = 7) plot.ts(seasonal_diff, main = "Seasonal Differenced (Lag 7) Box-Cox Transformed Series")

#Figure 3 generated by
adfTest(seasonal_diff, type = "c")

Step 6: ACF & PACF after Box-Cox + seasonal diff

#Figure 4 generated by
acf(seasonal_diff, main = "ACF of Box-Cox + Lag-7 Differenced Series") pacf(seasonal_diff, main = "PACF of Box-Cox + Lag-7 Differenced Series")

Step 7: Fit model using Box-Cox lambda and seasonal differencing internally

model <- auto.arima(seasonal_diff,trace=TRUE) summary(model) checkresiduals(model)

Step 8: Checking SARIMA Plots

#Figure 9 generated by
#a:SARIMA (1,0,1) (2,0,2) [7]

invisible(capture.output(sarima(seasonal_diff, p = 1, d = 0, q = 1, P = 2, D = 0, Q = 2, S = 7)))

#Figure 7 generated by
#b: SARIMA (2,0,0) (2, 0, 2) [7]

invisible(capture.output(sarima(seasonal_diff, p = 2, d = 0, q = 0, P=2,D=0,Q=2,S=7)))

Step 9: Fitting the final model to the original dataset and introducing D=1 to the original ts_data

ts_data <- ts(ts_data2$ride_count, frequency = 7) model_final <- Arima(ts_data, order = c(1, 0, 1), seasonal = list(order = c(2, 1, 2), period = 7), lambda = lambda)

Step 10: Checking the summary of both the models. match so that we can confirm orginal dataset well.

#Figure 10 generated by
summary(model)

#Figure 11 generated by
summary(model_final)

model_mid <- Arima(seasonal_diff, order = c(2, 0, 0), seasonal = list(order = c(2, 0, 2), period = 7))

#Figure 8 generated by
summary(model_mid)

Step 11: Forecast next 14 days

forecast_result <- forecast(model_final, h = 14)

#Figure 12 generated by
# Plotting the forecast with confidence interval
autoplot(forecast_result) + labs(title = "Forecasted Daily Ride Counts (July 1–14, 2024)", x = "Date", y = "Predicted Ride Count”) + scale_color_manual(values = c("blue", "red", "gray")) + theme_minimal(base_size = 14) + theme(plot.title = element_text(face = "bold", hjust = 0.5), legend.position = "bottom")

Step 12: Checking for Actual vs forecasted

a. Ensure date format

```
ts_data1$date <- as.Date(ts_data1$date)
```

b. Extract actual values for 1–14 July 2024

```
actual_july <- ts_data1[ts_data1$date >= as.Date("2024-07-01") & ts_data1$date <= as.Date("2024-07-14"), ]
```

c. Create forecast dataframe

```
forecast_df <- data.frame( date = seq(as.Date("2024-07-01"), by = "day", length.out = 14), forecast = as.numeric(forecast_result$mean) )
```

d.Merge actual and forecast

```
comparison <- merge(actual_july[, c("date", "ride_count")], forecast_df, by = "date", all = TRUE)
```

e. Plot

```
#Figure 13 generated by
library(ggplot2) ggplot(comparison, aes(x = date)) + geom_line(aes(y = ride_count, color = "Actual")) + geom_line(aes(y = forecast, color = "Forecast")) + labs(title = "Actual vs Forecasted Ride Counts (1–14 July 2024)", x = "Date", y = "Ride Count") + scale_color_manual(values = c("Actual" = "blue", "Forecast" = "red")) + theme_minimal()
```

f. Evaluting our model:

```
comparison_clean <-comparison mae <- mean(abs(comparison_clean$ride_count - comparison_clean$forecast)) rmse <- sqrt(mean((comparison_clean$ride_count - comparison_clean$forecast)^2)) mape <- mean(abs((comparison_clean$ride_count - comparison_clean$forecast) / comparison_clean$ride_count)) * 100 cat("Mean Absolute Error (MAE):", mae, "\n") cat("Root Mean Squared Error (RMSE):", rmse, "\n") cat("Mean Absolute Percentage Error (MAPE):", mape,"%")
```

2.4.2 Appendix 2: Code for the Data Aquisition:

```
library(dplyr)
library(lubridate)

Step 1: Define your folder path
folder_path <- "2023_2025" # Update this

Step 2: Read and combine CSVs (include member_casual)
csv_files <- list.files(path = folder_path, pattern = "\.csv$", full.names = TRUE)
combined_df <- lapply(csv_files, function(file) { df <- read.csv(file, stringsAsFactors = FALSE)

Check if all required columns are present
if (all(c("started_at", "ended_at", "member_casual") %in% names(df))) { df <- df[,
c("started_at", "ended_at", "member_casual")]
df$started_at <- as.character(df$started_at)
df$ended_at <- as.character(df$ended_at)

return(df)

} else { return(NULL) } }) %>% bind_rows()

Step 3: Parse datetime
combined_df$started_at <- ymd_hms(gsub("\..", "", combined_df$started_at))
combined_df$ended_at <- ymd_hms(gsub("\..", "", combined_df$ended_at))

Step 4: Fill missing started_at with ended_at
combined_df$started_at[is.na(combined_df$started_at)] <-
combined_df$ended_at[is.na(combined_df$started_at)]

Step 5: Keep started_at + member_casual, extract date/time
combined_df <- combined_df %>% mutate( date = as.Date(started_at), time = format(started_at,
"%H:%M") )

Step 6: Filter for member_casual = "Customer" and count rides per day
customer_data <- combined_df %>% filter(member_casual == "member") %>% group_by(date)
%>% summarise(ride_count = n(), .groups = "drop") %>% arrange(date)
Optional: Remove final row (if needed)
ts_data <- customer_data[-nrow(customer_data), ]

View the result
View(ts_data)
#write.csv(ts_data,"ts_data.csv",row.names=FALSE)
```