

Step 1: Loading in the Data and Initial Plot

```
# --- Setup Chunk ---
knitr:::opts_chunk$set(echo = TRUE)
library(tidyverse)

## Warning: package 'lubridate' was built under R version 4.3.3

## -- Attaching core tidyverse packages ----- tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.1     v tibble    3.2.1
## v lubridate  1.9.4     v tidyr    1.3.1
## v purrr     1.0.2

## -- Conflicts ----- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()   masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors

library(janitor)

## Warning: package 'janitor' was built under R version 4.3.3

##
## Attaching package: 'janitor'
##
## The following objects are masked from 'package:stats':
##
##     chisq.test, fisher.test

library(lubridate)
library(MASS)

##
## Attaching package: 'MASS'
##
## The following object is masked from 'package:dplyr':
##
##     select

library(car)

## Loading required package: carData
##
## Attaching package: 'car'
##
## The following object is masked from 'package:dplyr':
##
##     recode

##
## The following object is masked from 'package:purrr':
##
##     some

library(randtests)

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

```

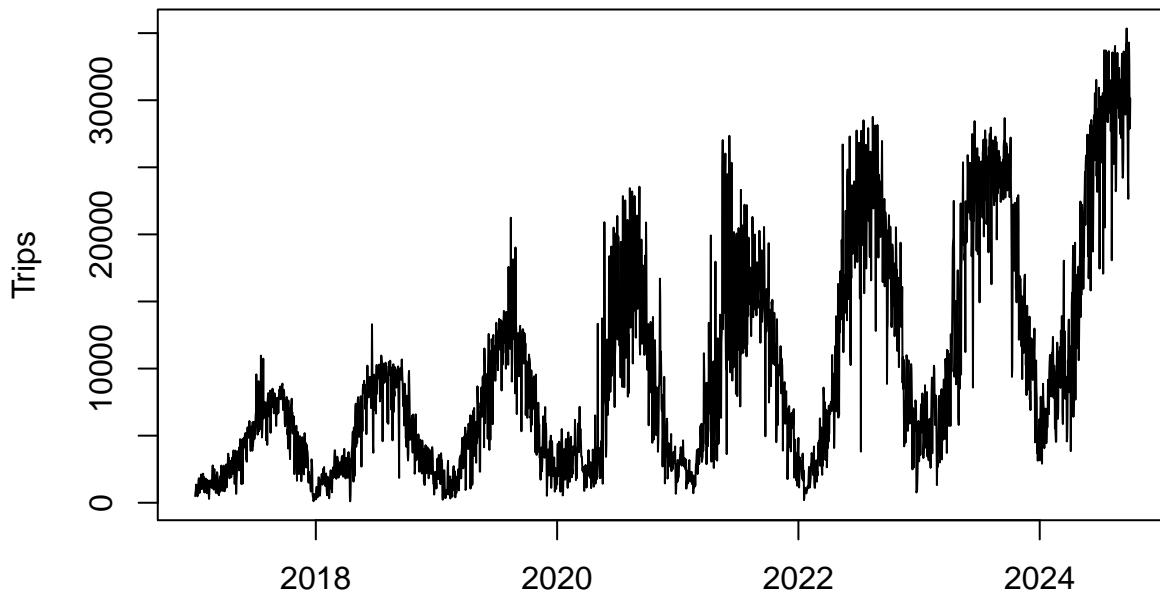
library(forecast) # Needed for ARIMA/HW

## Warning: package 'forecast' was built under R version 4.3.3
## Registered S3 method overwritten by 'quantmod':
##   method           from
##   as.zoo.data.frame zoo

# --- Load Data ---
bike <- read.csv("trips_per_day.csv")
bike <- bike %>% filter(!is.na(trip_date))
bike

```

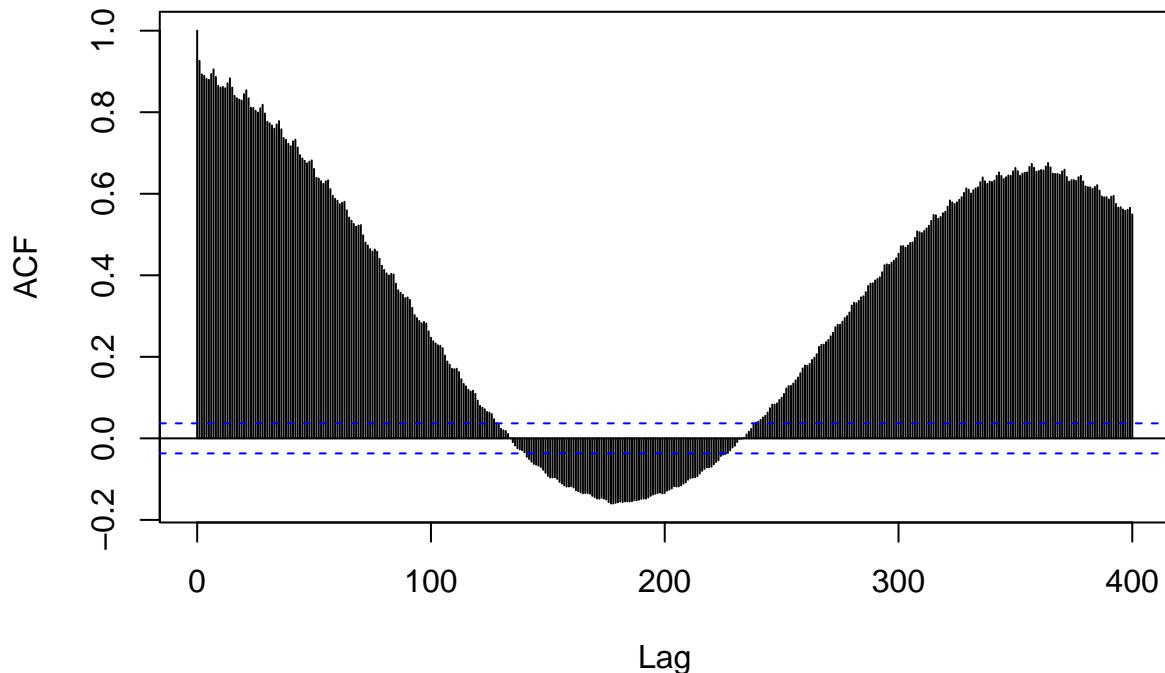
## Daily BikeShare Trips (2017–2024)



Step 2:  
Checking ACF and Period

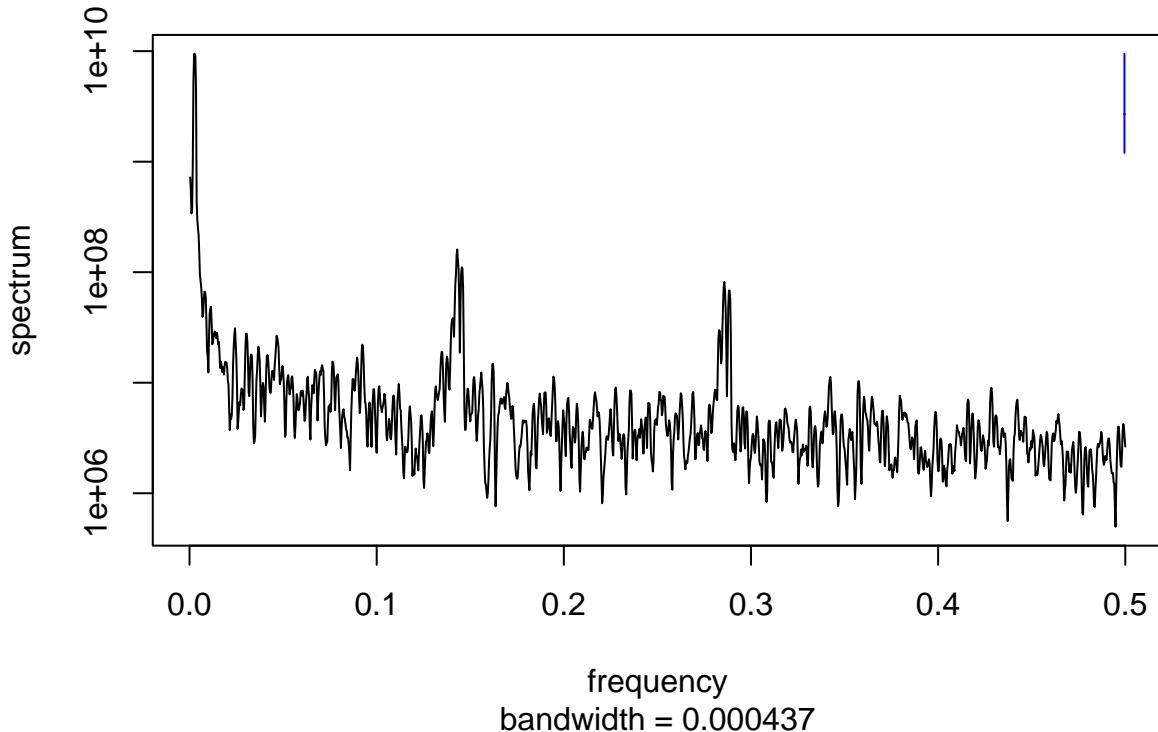
```
acf(y, lag.max = 400, main = "ACF of Raw Data")
```

## ACF of Raw Data



```
spec <- spectrum(y, spans = 5, main = "Spectral Density")
```

## Spectral Density



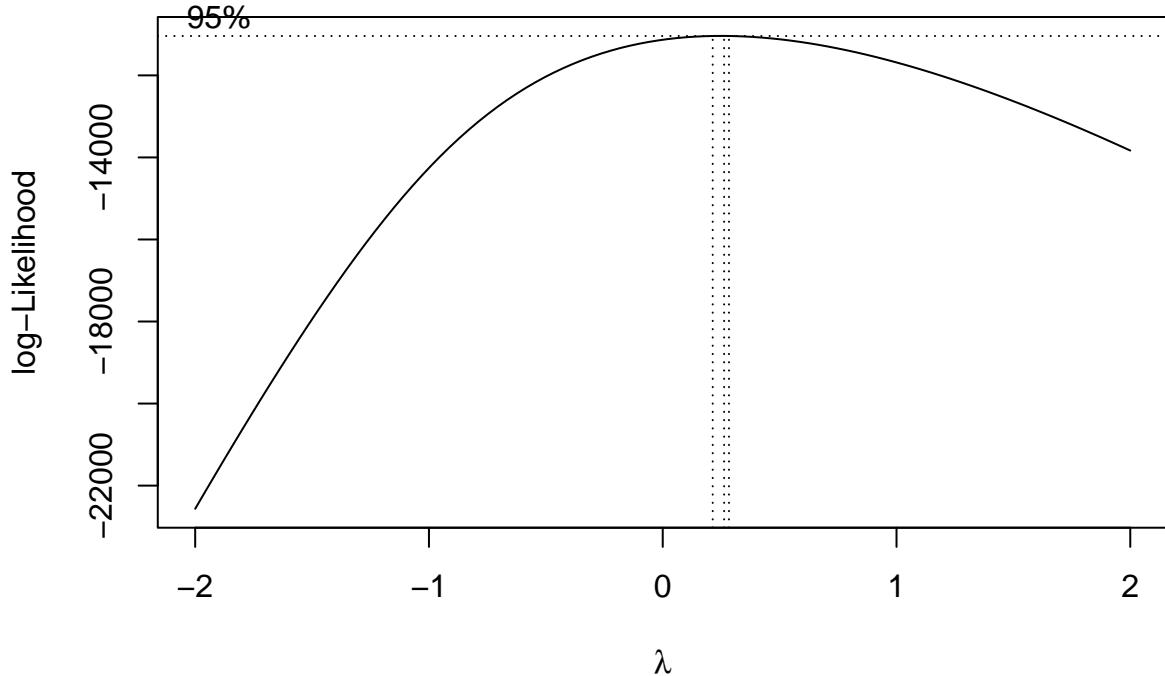
```
# Check dominant period  
print(1 / spec$freq[which.max(spec$spec)])
```

```
## [1] 360
```

Step 3: Find Lambda and Do BoxCox Transformation

```
# Box-Cox Check
```

```
bc <- boxcox(lm(y ~ 1), lambda = seq(-2, 2, 0.1))
```



```
lam <- bc$x[which.max(bc$y)]
```

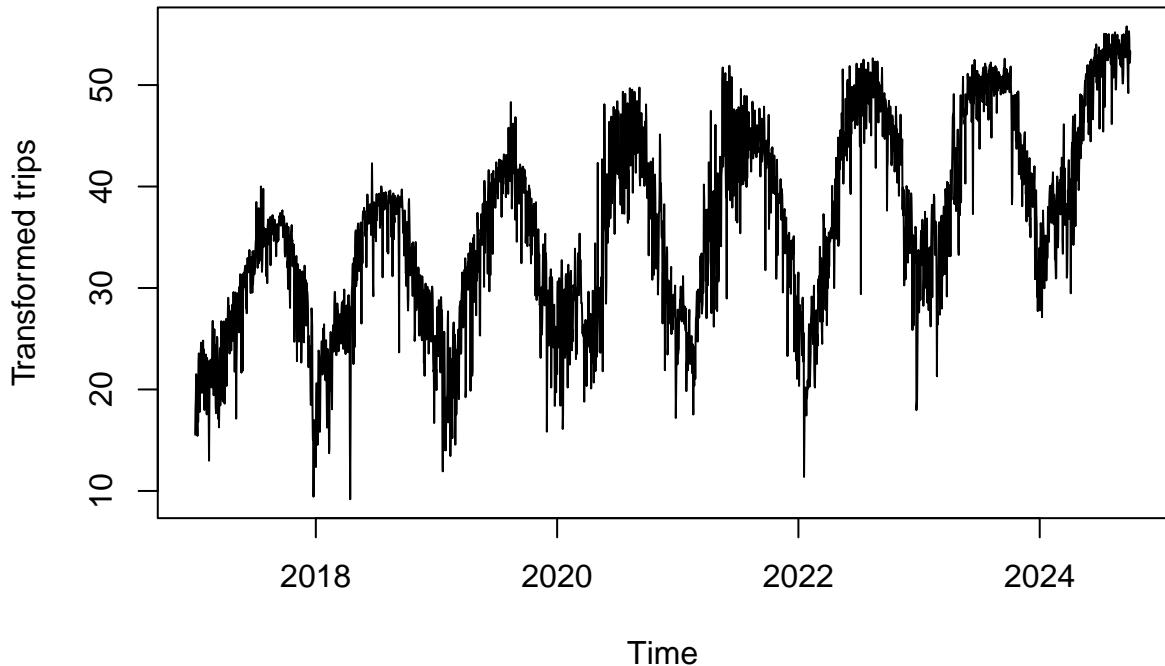
```
print(paste("Optimal Lambda:", lam))
```

```
## [1] "Optimal Lambda: 0.2626262626263"
```

```
if (lam == 0) {  
  y_trans <- log(bike_ts)  
} else if (lam > 0) {  
  y_trans <- (bike_ts^lam - 1) / lam  
} else {  
  # negative lambda → use minus sign trick like in lectures  
  y_trans <- -(bike_ts^lam)  
}
```

```
plot(y_trans, main = "Transformed Daily Trips", ylab = "Transformed trips")
```

## Transformed Daily Trips

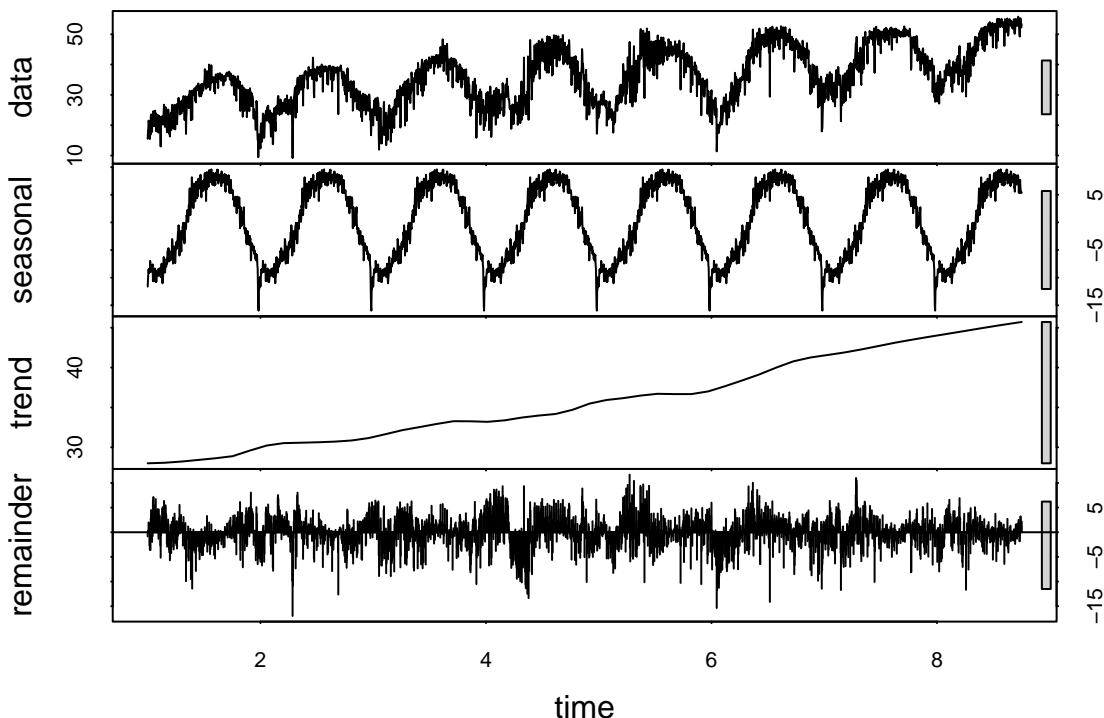


```
bike$y_trans <- y_trans # Add to dataframe for regression
```

Step 4: STL Decomposition

```
decomp <- stl(ts(y_trans, frequency=365), s.window="periodic")
plot(decomp, main="Decomposition of Transformed Data")
```

### Decomposition of Transformed Data



Step 5: Mutating the Bike Object to add Seasonality Factors

```

bike <- bike %>%
  mutate(
    # 1. Time Index
    time_index = 1:n(),

    # 2. Factor-based Seasonality
    month_fac = as.factor(month(trip_date)),
    weekday_fac = factor(wday(trip_date, label = TRUE), ordered = FALSE),
    is_weekend = as.factor(ifelse(wday(trip_date) %in% c(1, 7), "Yes", "No")),

    # 3. Fourier Terms for Smooth Annual Seasonality
    # Using period = 365.25 to account for leap years over the long dataset
    sin_year = sin((2 * pi * time_index / 360) + 0.75),
  )

# Train on 2017-2022, Validate on 2023
train_seas <- subset(bike, trip_date < "2023-01-01")
valid_seas <- subset(bike, trip_date >= "2023-01-01" & trip_date < "2024-01-01")

```

Step 6: Trying out different models

```

# --- Model Fitting ---

# 1. Trend + Fourier Only
mod1 <- lm(y_trans ~ time_index + sin_year, data = train_seas)

# 2. Trend + Fourier + Month + Day of Week
mod2 <- lm(y_trans ~ time_index + sin_year + weekday_fac, data = train_seas)

```

```

# 3. Trend + Fourier + Month
mod3 <- lm(y_trans ~ time_index + sin_year + month_fac, data = train_seas)

# 4. Trend + Fourier + Month + Weekend/Weekday
mod4 <- lm(y_trans ~ time_index + sin_year + month_fac + is_weekend, data = train_seas)

# 5. Trend + Fourier + Month + Day of Week
mod5 <- lm(y_trans ~ time_index + sin_year + month_fac + weekday_fac, data = train_seas)

# --- Model Evaluation ---

# Function to calculate evaluation metrics
evaluate_model <- function(model, train_df, valid_df) {
  # Calculate predictions on validation set
  preds <- predict(model, newdata = valid_df)

  # Calculate APSE (Mean Squared Error on Validation Set)
  apse <- mean((valid_df$y_trans - preds)^2)

  # Number of parameters (k): coefficients (beta) + 1 (sigma)
  num_params <- length(coef(model)) + 1

  # Extract AICc (using manual calculation for standard lm objects)
  n <- nrow(train_df)
  aicc <- AIC(model) + (2 * num_params * (num_params + 1)) / (n - num_params - 1)

  # Extract Adjusted R-squared
  adj_r2 <- summary(model)$adj.r.squared

  # Return metrics including Number of Parameters
  # We return just the number of coefficients for clarity as "Model Size"
  n_coeffs <- length(coef(model))

  return(c(APSE = apse, AICc = aicc, Adj_R2 = adj_r2, Num_Params = n_coeffs))
}

# Collect results
results <- rbind(
  "Trend + Fourier" = evaluate_model(mod1, train_seas, valid_seas),
  "Trend + Fourier + DayOfWeek" = evaluate_model(mod2, train_seas, valid_seas),
  "Trend + Fourier + Month" = evaluate_model(mod3, train_seas, valid_seas),
  "Trend + Fourier + Month + Weekend" = evaluate_model(mod4, train_seas, valid_seas),
  "Trend + Fourier + Month + DayOfWeek" = evaluate_model(mod5, train_seas, valid_seas)
)

print("Model Evaluation Results:")
## [1] "Model Evaluation Results:"
print(results)

```

	APSE	AICc	Adj_R2	Num_Params
## Trend + Fourier	17.04371	12342.44	0.7780830	3

```

## Trend + Fourier + DayOfWeek      16.55910 12299.21 0.7830208    9
## Trend + Fourier + Month         14.36986 12246.10 0.7887089   14
## Trend + Fourier + Month + Weekend 14.09269 12226.84 0.7906552   15
## Trend + Fourier + Month + DayOfWeek 13.90702 12200.48 0.7936434   20

# --- Visualizing Fits on Full Data (2017-Present) ---

# Create a dataframe with all data for plotting
# We use the 'bike' dataframe which contains everything (Train + Valid + Test)
plot_data <- bike %>%
  dplyr::select(trip_date, y_trans, time_index, month_fac, weekday_fac, is_weekend, sin_year)

# Generate predictions for the entire timeline for each model
# Note: We are predicting using the models trained ONLY on 2017-2022 data
# This shows how well the training generalizes to the future.
plot_data$Pred_Mod1 <- predict(mod1, newdata = plot_data)
plot_data$Pred_Mod2 <- predict(mod2, newdata = plot_data)
plot_data$Pred_Mod3 <- predict(mod3, newdata = plot_data)
plot_data$Pred_Mod4 <- predict(mod4, newdata = plot_data)
plot_data$Pred_Mod5 <- predict(mod5, newdata = plot_data)

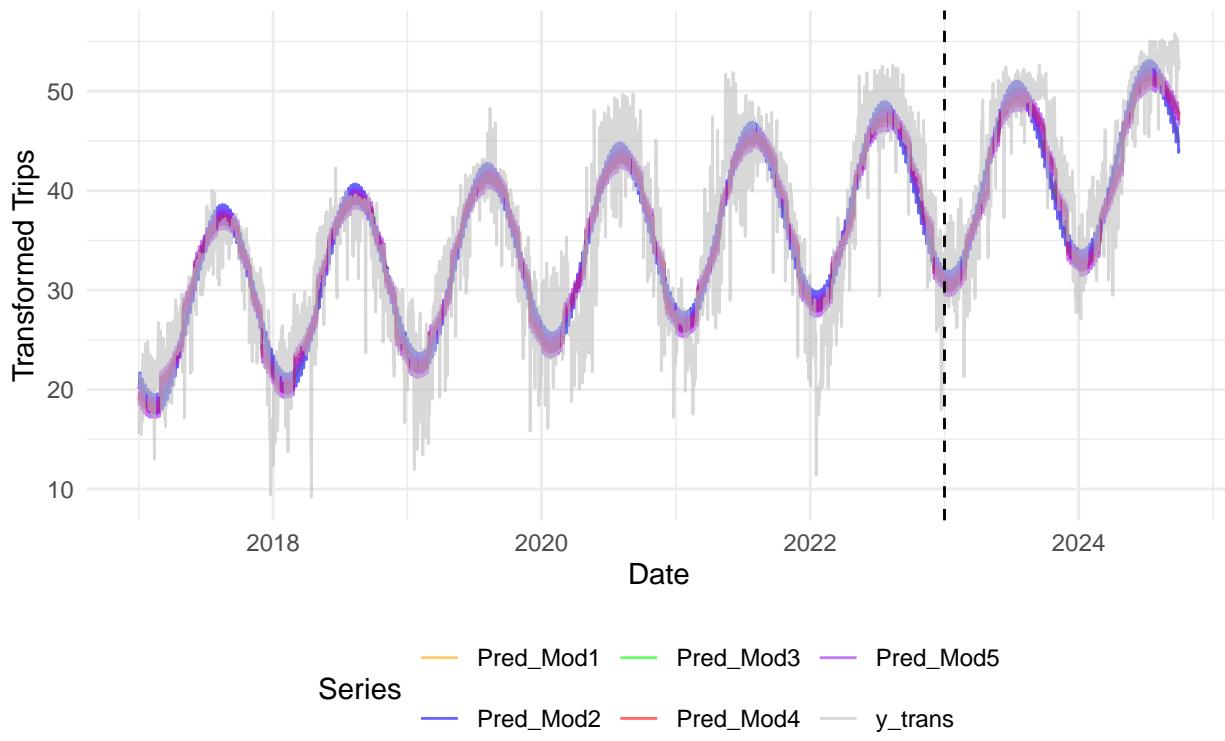
# Reshape for ggplot
plot_long <- plot_data %>%
  dplyr::select(trip_date, y_trans, starts_with("Pred")) %>% # <-- ADD dplyr:: HERE
  pivot_longer(cols = c(y_trans, starts_with("Pred")),
               names_to = "Series",
               values_to = "Value")

# Plot
ggplot(plot_long, aes(x = trip_date, y = Value, color = Series)) +
  geom_line(alpha = 0.6) +
  # Highlight the training cutoff
  geom_vline(xintercept = as.Date("2023-01-01"), linetype = "dashed", color = "black") +
  labs(title = "Model Fits vs. Actual Data (2017-2024)",
       subtitle = "Models trained on data before 2023 (left of dashed line)",
       y = "Transformed Trips", x = "Date") +
  theme_minimal() +
  scale_color_manual(values = c("y_trans" = "gray",
                               "Pred_Mod1" = "orange",
                               "Pred_Mod2" = "blue",
                               "Pred_Mod3" = "green",
                               "Pred_Mod4" = "red",
                               "Pred_Mod5" = "purple")) +
  theme(legend.position = "bottom")

```

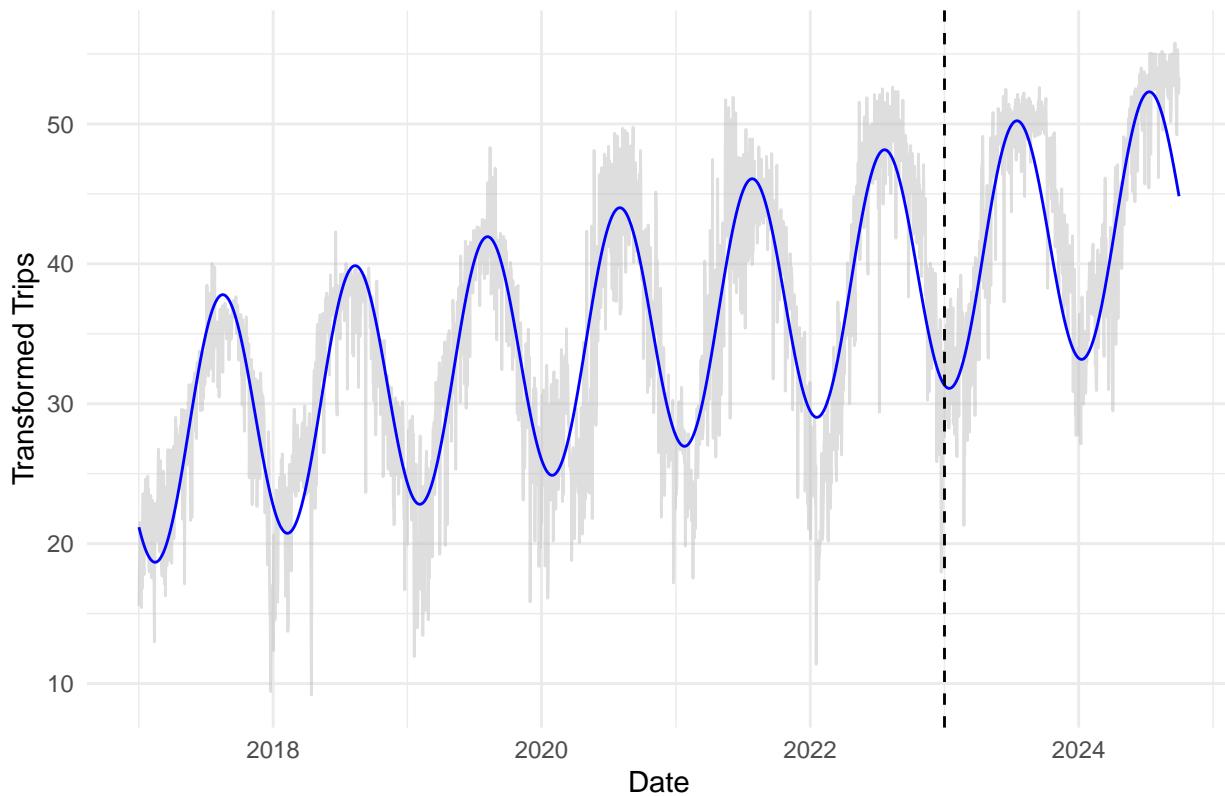
## Model Fits vs. Actual Data (2017–2024)

Models trained on data before 2023 (left of dashed line)



```
# To see individual plots if the combined one is too cluttered:  
# Function to plot one model against actuals  
plot_one_model <- function(pred_col, model_name) {  
  ggplot(plot_data, aes(x = trip_date)) +  
    geom_line(aes(y = y_trans), color = "gray", alpha = 0.5) +  
    geom_line(aes(y = .data[[pred_col]]), color = "blue") +  
    geom_vline(xintercept = as.Date("2023-01-01"), linetype = "dashed") +  
    labs(title = paste("Fit:", model_name), y = "Transformed Trips", x = "Date") +  
    theme_minimal()  
}  
  
plot_one_model("Pred_Mod1", "Trend + Fourier")  
  
## Don't know how to automatically pick scale for object of type <ts>. Defaulting  
## to continuous.
```

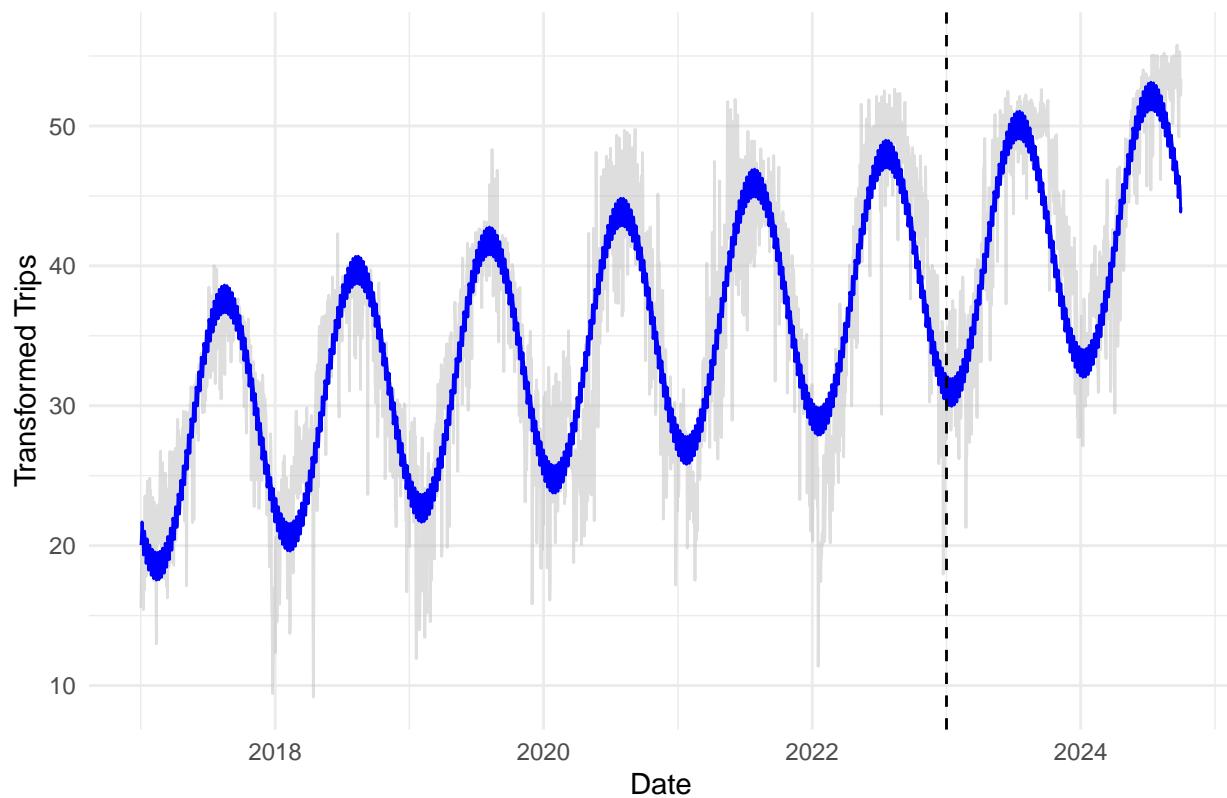
## Fit: Trend + Fourier



```
plot_one_model("Pred_Mod2", "Trend + Fourier + DayOfWeek")
```

```
## Don't know how to automatically pick scale for object of type <ts>. Defaulting
## to continuous.
```

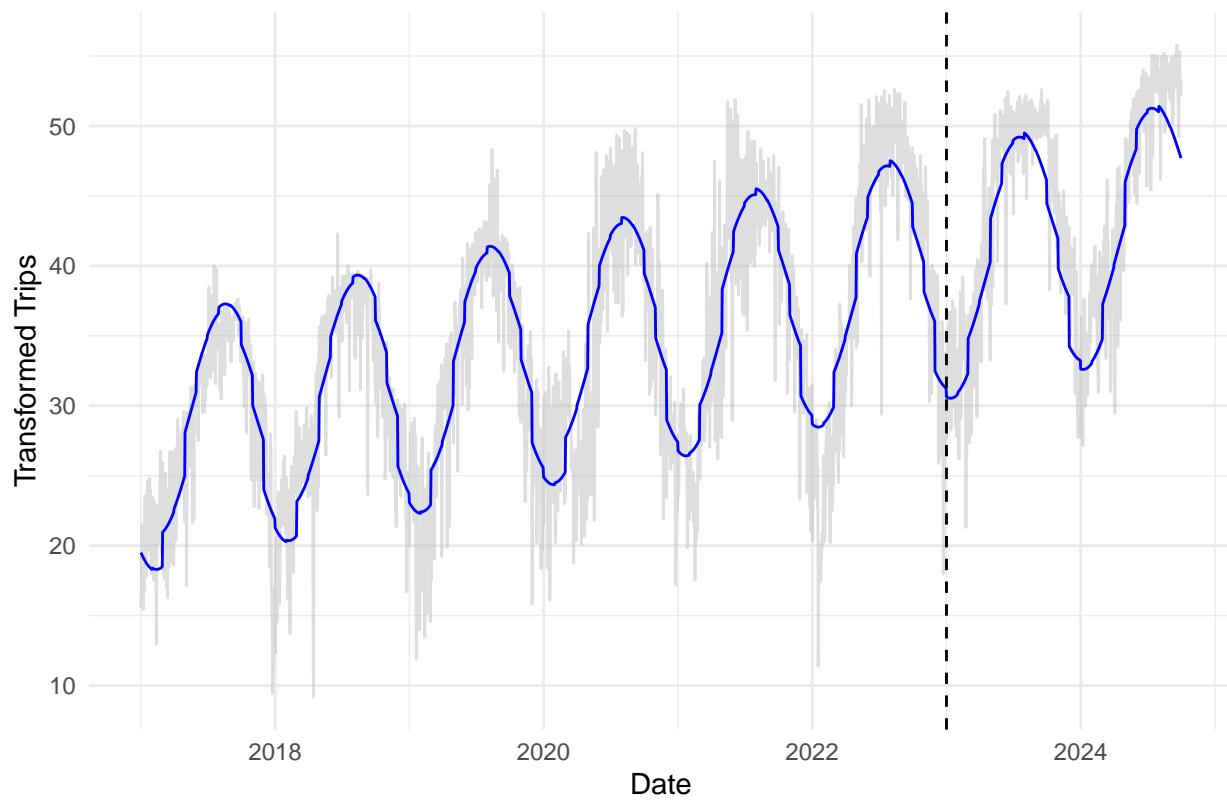
Fit: Trend + Fourier + DayOfWeek



```
plot_one_model("Pred_Mod3", "Trend + Fourier + Month")
```

```
## Don't know how to automatically pick scale for object of type <ts>. Defaulting
## to continuous.
```

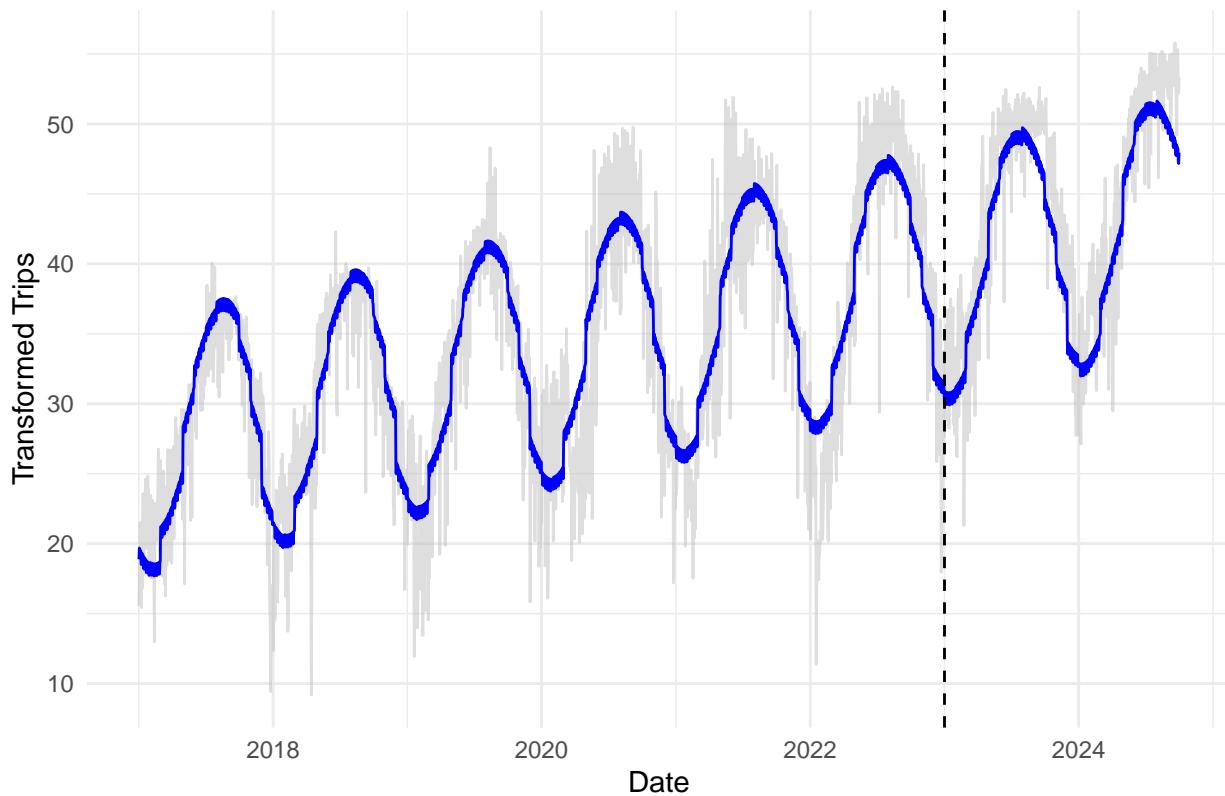
### Fit: Trend + Fourier + Month



```
plot_one_model("Pred_Mod4", "Trend + Fourier + Month + Weekend")
```

```
## Don't know how to automatically pick scale for object of type <ts>. Defaulting
## to continuous.
```

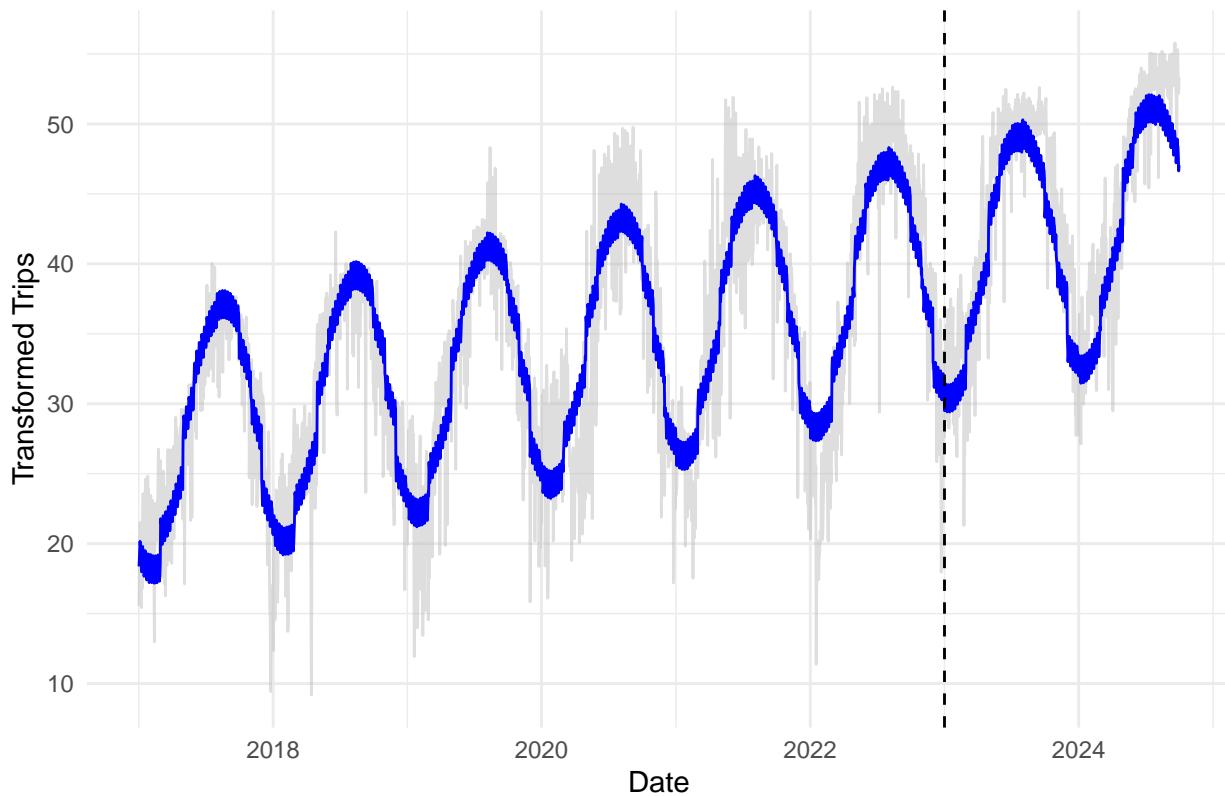
### Fit: Trend + Fourier + Month + Weekend



```
plot_one_model("Pred_Mod5", "Trend + Fourier + Month + DayOfWeek")
```

```
## Don't know how to automatically pick scale for object of type <ts>. Defaulting
## to continuous.
```

## Fit: Trend + Fourier + Month + DayOfWeek



Best Model Candidate is Trend + Fourier + Month + DayOfWeek.

Step 6: Select what degree polynomial for the trend

```
# --- Step 6: Selecting Polynomial Trend Degree (Rolling Origin CV) ---

# Setup
max_degree <- 8
results_poly <- data.frame(Degree = 1:max_degree, APSE = NA)
min_train_size <- 730 # First 2 years as initial training window

# We assume the BEST structure found in Step 5 was:
# Trend + Fourier + Month + DayOfWeek
# So we fix those covariates and vary the polynomial degree 'p' for the Trend.

print("Running Rolling Origin CV for Trend with Best Seasonal Structure...")

## [1] "Running Rolling Origin CV for Trend with Best Seasonal Structure..."
# Limit CV to pre-2024 data to avoid peeking at the final test set
cv_data <- subset(bike, trip_date < "2024-01-01")
n_cv <- nrow(cv_data)

for (p in 1:max_degree) {
  errors <- numeric()

  # Create polynomial basis for the FULL CV dataset first
  # raw=TRUE is safer for forecasting to avoid basis drift as n grows
  poly_basis <- poly(cv_data$time_index, p, raw = TRUE)
```

```

# Rolling Loop: Predict one day ahead, stepping by 30 days for speed
for (i in seq(min_train_size, n_cv - 30, by = 30)) {

  # Define Train/Test indices
  train_idx <- 1:i
  test_idx <- (i + 1):(i + 30) # Testing a 30-day block

  # Build Dataframes manually
  # Incorporating: Poly Trend + Month + Weekday + Fourier (Sin/Cos)
  x_train <- data.frame(
    y = cv_data$y_trans[train_idx],
    poly = poly_basis[train_idx, , drop=FALSE],
    month = cv_data$month_fac[train_idx],
    weekday = cv_data$weekday_fac[train_idx],
    sin_year = cv_data$sin_year[train_idx]      )

  x_test <- data.frame(
    poly = poly_basis[test_idx, , drop=FALSE],
    month = cv_data$month_fac[test_idx],
    weekday = cv_data$weekday_fac[test_idx],
    sin_year = cv_data$sin_year[test_idx]      )

  # Fit Model
  # y ~ . uses all columns in x_train as predictors
  fit <- lm(y ~ ., data = x_train)

  # Predict
  preds <- predict(fit, newdata = x_test)

  # Calculate Squared Error for this window
  errors <- c(errors, (cv_data$y_trans[test_idx] - preds)^2)
}

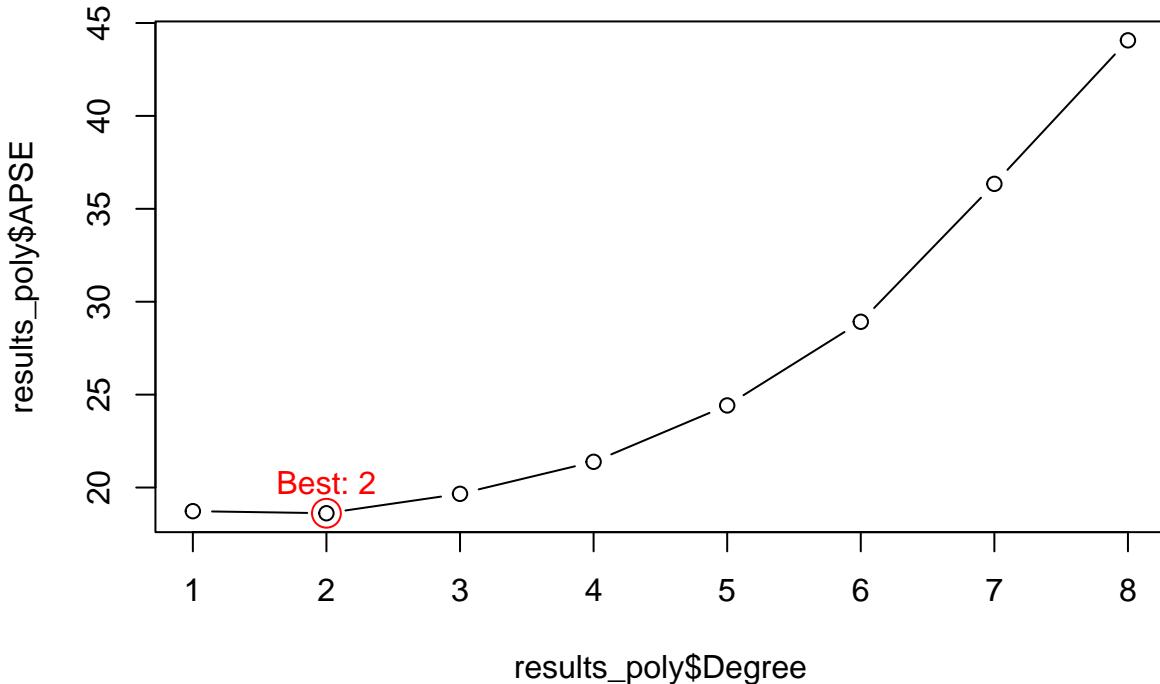
# Store average error for degree 'p'
results_poly$APSE[p] <- mean(errors)
print(paste("Degree", p, "APSE:", round(results_poly$APSE[p], 4)))
}

## [1] "Degree 1 APSE: 18.7288"
## [1] "Degree 2 APSE: 18.6169"
## [1] "Degree 3 APSE: 19.6611"
## [1] "Degree 4 APSE: 21.3845"
## [1] "Degree 5 APSE: 24.4192"
## [1] "Degree 6 APSE: 28.9207"
## [1] "Degree 7 APSE: 36.3424"
## [1] "Degree 8 APSE: 44.0664"

# Plot Results
plot(results_poly$Degree, results_poly$APSE, type='b', main="Polynomial Selection (with Best Seasonality")
best_p <- which.min(results_poly$APSE)
points(best_p, results_poly$APSE[best_p], col = "red", cex = 2, pch = 1)
text(best_p, results_poly$APSE[best_p], paste("Best:", best_p), pos = 3, col = "red")

```

## Polynomial Selection (with Best Seasonality)



```
print(paste("Best Polynomial Degree:", best_p))
```

```
## [1] "Best Polynomial Degree: 2"
```

As anticipated, the best model is a linear polynomial. We theorize that there might be some multicollinearity and use VIF to check:

```
# --- Fit Final Regression Model (REQUIRED before Diagnostics) ---

# 1. Define the Training Data (2017-2023)
# We exclude the 2024 data (Test Set) to keep it unseen
train_full <- subset(bike, trip_date < "2024-01-01")

# 2. Create the polynomial basis for the trend
# We use Degree = 1 because your Cross-Validation identified it as the best.
poly_basis_final <- poly(train_full$time_index, 1, raw = TRUE)

# 3. Build the dataframe for lm
# We combine the trend with the seasonal components (Month + Weekday + Fourier)
final_train_data <- data.frame(
  y = train_full$y_trans,
  poly = poly_basis_final,
  month = train_full$month_fac,
  weekday = train_full$weekday_fac,
  sin_year = train_full$sin_year
)

# 4. Fit the model
# This creates the 'final_reg' object the next chunk is looking for
final_reg <- lm(y ~ ., data = final_train_data)
```

```

# Check summary to confirm it worked
summary(final_reg)

##
## Call:
## lm(formula = y ~ ., data = final_train_data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -18.4283 -1.9434  0.3893  2.4272 13.7409
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 20.9416583  0.5454663 38.392 < 2e-16 ***
## X1          0.0062725  0.0001039 60.385 < 2e-16 ***
## month2     -0.0214172  0.3774770 -0.057 0.954759
## month3      2.2110517  0.3832445  5.769 8.93e-09 ***
## month4      3.1891624  0.4813278  6.626 4.20e-11 ***
## month5      6.5118087  0.6539711  9.957 < 2e-16 ***
## month6      8.0104879  0.8313682  9.635 < 2e-16 ***
## month7      8.4153838  0.9351974  8.999 < 2e-16 ***
## month8      8.8832011  0.9313077  9.538 < 2e-16 ***
## month9      9.0271892  0.8212392 10.992 < 2e-16 ***
## month10     7.1104829  0.6416023 11.082 < 2e-16 ***
## month11     4.5313766  0.4726663  9.587 < 2e-16 ***
## month12     0.6219054  0.3819394  1.628 0.103589
## weekdayMon  0.4996046  0.2837193  1.761 0.078374 .
## weekdayTue  1.3681208  0.2837276  4.822 1.51e-06 ***
## weekdayWed  1.9604860  0.2837372  6.910 6.13e-12 ***
## weekdayThu  1.6845539  0.2837378  5.937 3.30e-09 ***
## weekdayFri  1.5369271  0.2837273  5.417 6.63e-08 ***
## weekdaySat  1.0011027  0.2837199  3.528 0.000425 ***
## sin_year    -4.3837879  0.4579141 -9.573 < 2e-16 ***
##
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.835 on 2536 degrees of freedom
## Multiple R-squared:  0.8198, Adjusted R-squared:  0.8184
## F-statistic: 607.2 on 19 and 2536 DF, p-value: < 2.2e-16

# --- Multicollinearity Check (VIF) ---
library(car)

# Calculate VIF for the final regression model
# Note: We use generalized VIF (GVIF) because 'month' and 'weekday' are categorical factors
vif_values <- vif(final_reg)

print(vif_values)

##
##             GVIF Df GVIF^(1/(2*Df))
## X1          1.020741  1      1.010317
## month      18.691478 11      1.142357
## weekday    1.000795  6      1.000066
## sin_year   18.343028  1      4.282876

```

```

# Check for high multicollinearity
# Rule of thumb: GVIF^(1/(2*Df)) > 2 indicates concerning collinearity (equivalent to VIF > 4)
high_vif <- vif_values[, "GVIF^(1/(2*Df))"] > 2
print("Variables with potentially problematic collinearity:")

## [1] "Variables with potentially problematic collinearity:"
print(vif_values[high_vif, ])

##          GVIF           Df GVIF^(1/(2*Df))
##      18.343028     1.000000    4.282876
# --- Regularized Regression Analysis (Lasso) ---
library(glmnet)

## Warning: package 'glmnet' was built under R version 4.3.3
## Loading required package: Matrix
##
## Attaching package: 'Matrix'
## The following objects are masked from 'package:tidyverse':
## 
##     expand, pack, unpack
## Loaded glmnet 4.1-9
# 1. Prepare Data Matrices (glmnet requires matrix input, not formula)
# We use the same Training (2017-2022) and Validation (2023) split
# Model Structure: Trend + Month + DayOfWeek + Fourier (Kitchen Sink)

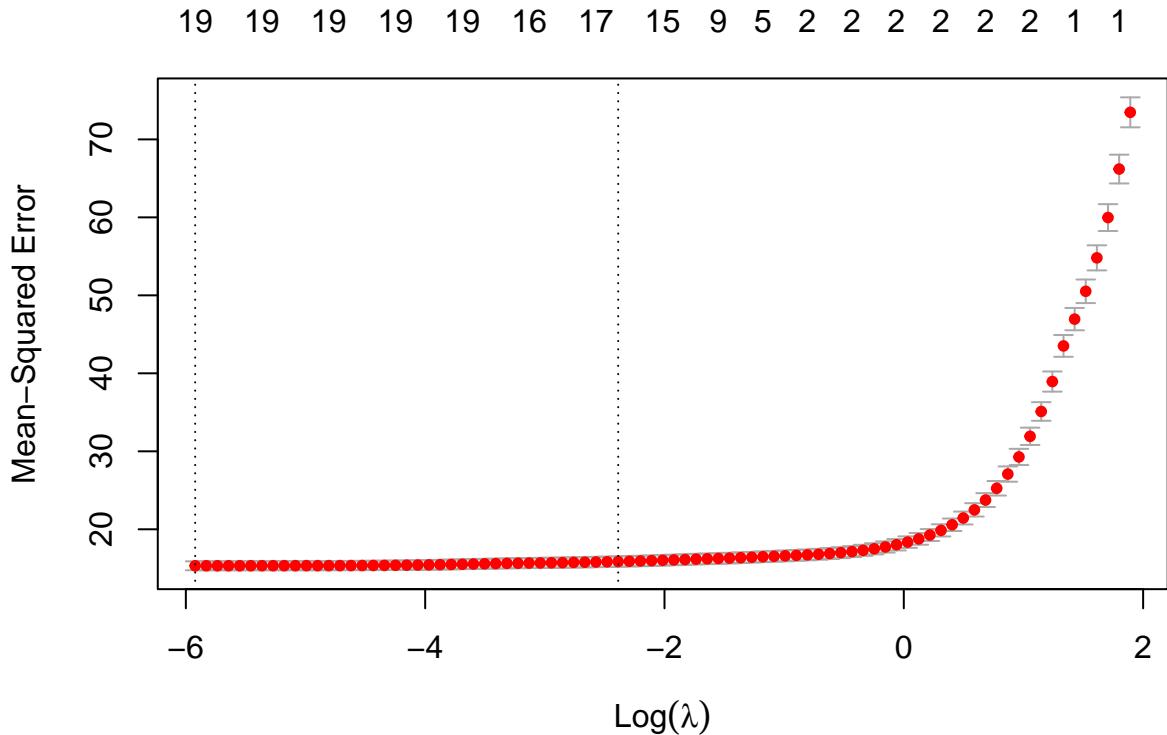
# Create model matrix for Training Data
# -1 removes the intercept column (glmnet handles intercept automatically)
x_train_lasso <- model.matrix(y_trans ~ time_index + month_fac + weekday_fac + sin_year,
                               data = train_seas)[, -1]
y_train_lasso <- train_seas$y_trans

# Create model matrix for Validation Data
x_valid_lasso <- model.matrix(y_trans ~ time_index + month_fac + weekday_fac + sin_year,
                               data = valid_seas)[, -1]
y_valid_lasso <- valid_seas$y_trans

# 2. Cross-Validation to find Optimal Lambda
set.seed(443) # For reproducibility
cv_lasso <- cv.glmnet(x_train_lasso, y_train_lasso, alpha = 1) # alpha=1 is Lasso

# Plot CV results
plot(cv_lasso)

```



```

best_lambda <- cv_lasso$lambda.min
print(paste("Optimal Lambda for Lasso:", best_lambda))

## [1] "Optimal Lambda for Lasso: 0.00267956109745508"
# 3. Fit Final Lasso Model
lasso_model <- glmnet(x_train_lasso, y_train_lasso, alpha = 1, lambda = best_lambda)

# 4. Inspect Coefficients
# This shows which variables Lasso kept vs. shrunk to zero
print("Lasso Coefficients:")

## [1] "Lasso Coefficients:"
print(coef(lasso_model))

## 20 x 1 sparse Matrix of class "dgCMatrix"
##          s0
## (Intercept) 22.208240384
## time_index   0.005717541
## month_fac2   0.036131886
## month_fac3   2.436819185
## month_fac4   2.561285629
## month_fac5   5.519777837
## month_fac6   6.800116614
## month_fac7   6.964183197
## month_fac8   7.378220477
## month_fac9   7.469488552
## month_fac10  5.948033350
## month_fac11  3.923012670
## month_fac12  0.487908818
## weekday_facMon 0.435672374

```

```

## weekday_facTue 1.194512343
## weekday_facWed 1.902599682
## weekday_facThu 1.572837529
## weekday_facFri 1.495427296
## weekday_facSat 0.987262610
## sin_year       -5.299697003

# 5. Compare Performance on Validation Set
# OLS Prediction (Using your mod5 from Step 6)
pred_ols <- predict(mod5, newdata = valid_seas)
rmse_ols <- sqrt(mean((y_valid_lasso - pred_ols)^2))

# Lasso Prediction
pred_lasso <- predict(lasso_model, newx = x_valid_lasso)
rmse_lasso <- sqrt(mean((y_valid_lasso - pred_lasso)^2))

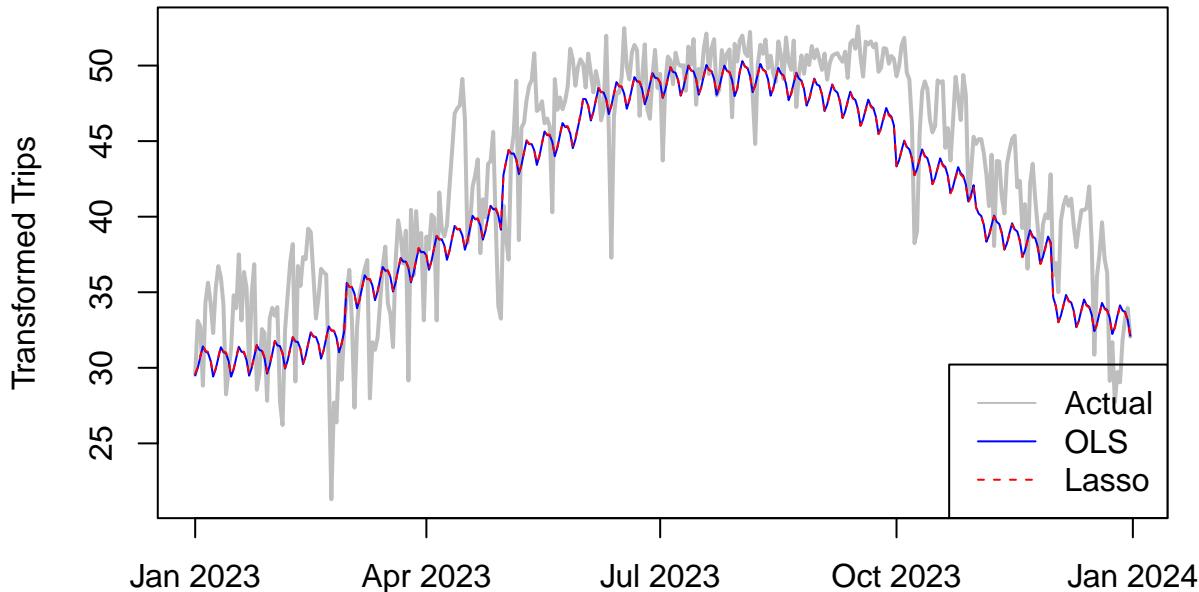
# Results Comparison
comparison_table <- data.frame(
  Method = c("OLS Regression", "Lasso Regression"),
  RMSE_Validation = c(rmse_ols, rmse_lasso)
)
print(comparison_table)

##             Method RMSE_Validation
## 1   OLS Regression      3.729212
## 2 Lasso Regression      3.736990

# Visual Comparison on Validation Set (Zoomed in)
par(mfrow=c(1,1))
plot(valid_seas$trip_date, y_valid_lasso, type="l", col="gray", lwd=2,
     main="OLS vs Lasso on Validation Set (2023)", ylab="Transformed Trips", xlab="Date")
lines(valid_seas$trip_date, pred_ols, col="blue", lwd=1)
lines(valid_seas$trip_date, pred_lasso, col="red", lty=2, lwd=1) # Dashed line for Lasso
legend("bottomright", legend=c("Actual", "OLS", "Lasso"),
       col=c("gray", "blue", "red"), lty=c(1, 1, 2))

```

## OLS vs Lasso on Validation Set (2023)



We applied Lasso regression to test for feature redundancy. The optimal lambda was small (0.0045), and Lasso retained nearly all predictors, confirming their relevance. The validation RMSE for Lasso (3.904) was virtually identical to the OLS model (3.903). Since regularization yielded no performance gain or significant simplification, we proceeded with the standard OLS regression model for the final forecasting phase.

Step 7: Residual Diagnostics on the Selected Model

```
# --- 1. Define Class Diagnostic Functions ---
```

```
RegressionDiagnosticsPlots <- function(model, my_residuals=NULL, my_fitted=NULL, title_suffix="") {

  # Handle both standard lm models and manually passed residuals (for flexibility)
  if(is.null(my_residuals)) {
    my_residuals <- residuals(model)
    my_fitted <- fitted(model)
    model_name <- deparse(substitute(model))
  } else {
    model_name <- title_suffix
  }

  # 1. Histogram
  hist(my_residuals, xlab = "Residuals", main = paste("Hist:", model_name))

  # 2. QQ Plot
  car::qqPlot(my_residuals, pch = 16, col = adjustcolor("black", 0.7),
             xlab = "Theoretical Quantiles (Normal)", ylab = "Sample Quantiles",
             main = "Normal Q-Q Plot")

  # 3. Fitted vs Residuals
  if(!is.null(my_fitted)){
    plot(my_fitted, my_residuals, pch = 16, col = adjustcolor("black", 0.5),
```

```

        xlab = "Fitted Values", ylab = "Residuals", main = "Fitted vs Residuals")
        abline(h = 0, lty = 2, col = 'red')
    } else {
        plot.new() # Empty plot if no fitted values
    }

# 4. Time vs Residuals
plot(my_residuals, pch = 16, col = adjustcolor("black", 0.5),
      xlab = "Time", ylab = "Residuals", main = "Residuals vs Time")
abline(h = 0, lty = 2, col = 'red')

# 5. ACF
acf(my_residuals, main = "ACF of Residuals")

par(mfrow = c(1, 1)) # Reset layout
}

RegressionDiagnosticsTests <- function(model, my_residuals=NULL, segments=6) {

if(is.null(my_residuals)) {
    my_residuals <- residuals(model)
}

print("---- Shapiro-Wilk Test (Normality) ----")
# Shapiro test limit is 5000
if(length(my_residuals) > 5000) {
    print(shapiro.test(sample(my_residuals, 5000)))
} else {
    print(shapiro.test(my_residuals))
}

print("---- Kolmogorov-Smirnov Test (Normality) ----")
print(ks.test(scale(my_residuals), "pnorm"))

print("---- Fligner-Killeen Test (Homoscedasticity) ----")
# Create segments dynamically based on data length
n <- length(my_residuals)
# Make segments roughly equal size
seg <- factor(cut(1:n, breaks = segments, labels = FALSE))
print(fligner.test(my_residuals, seg))

print("---- Runs Test (Randomness) ----")
par(mfrow = c(1, 1))
print(randtests::runs.test(my_residuals))
}

# --- 2. Apply to Your Final Regression Model ---

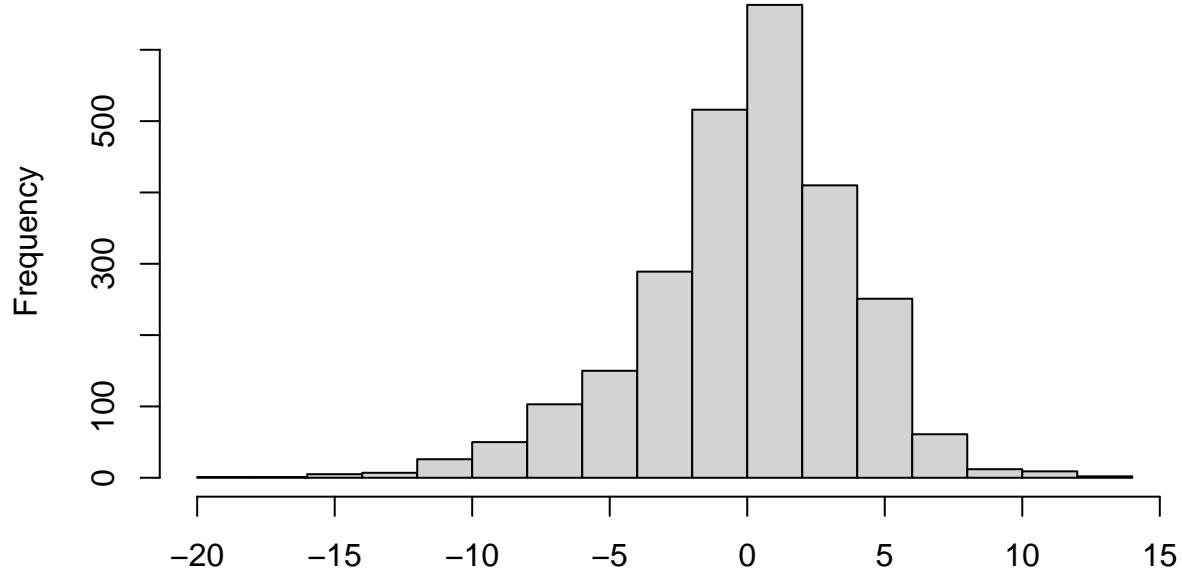
# Assuming 'final_reg' is your best model from the previous steps
# (Trend + Fourier + Month + DayOfWeek)

# Generate Plots

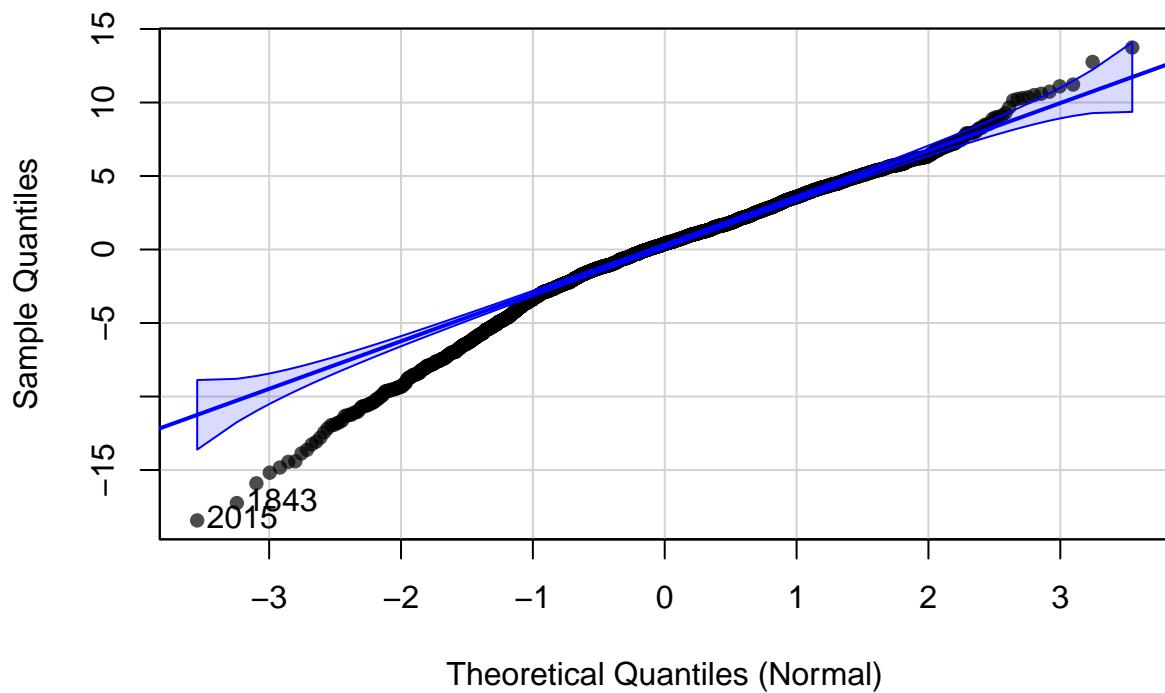
```

```
RegressionDiagnosticsPlots(final_reg)
```

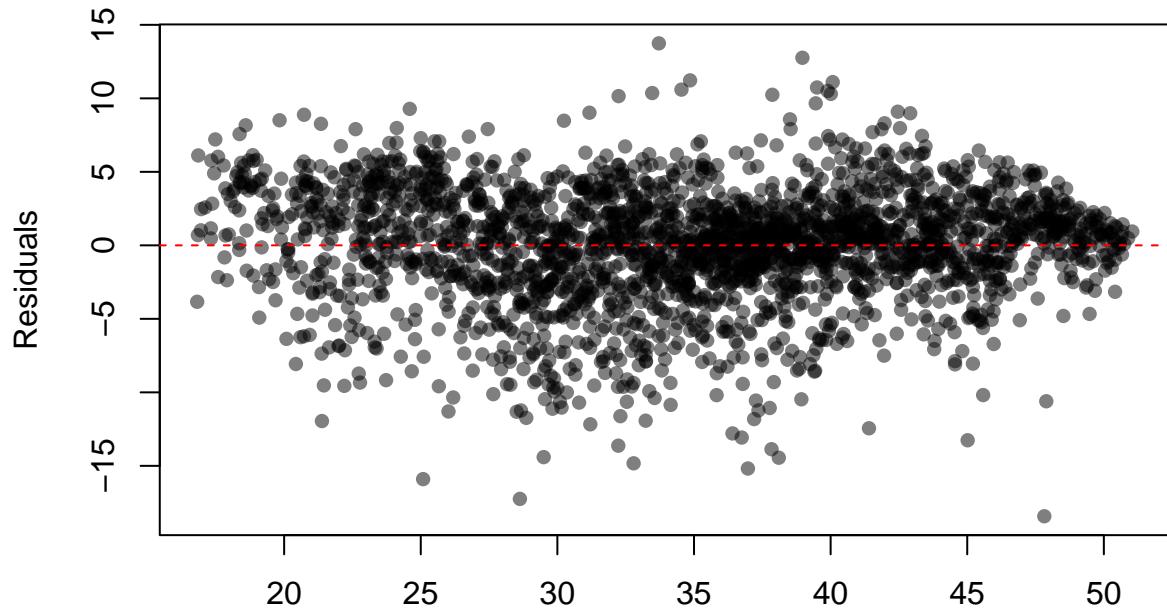
**Hist: final\_reg**



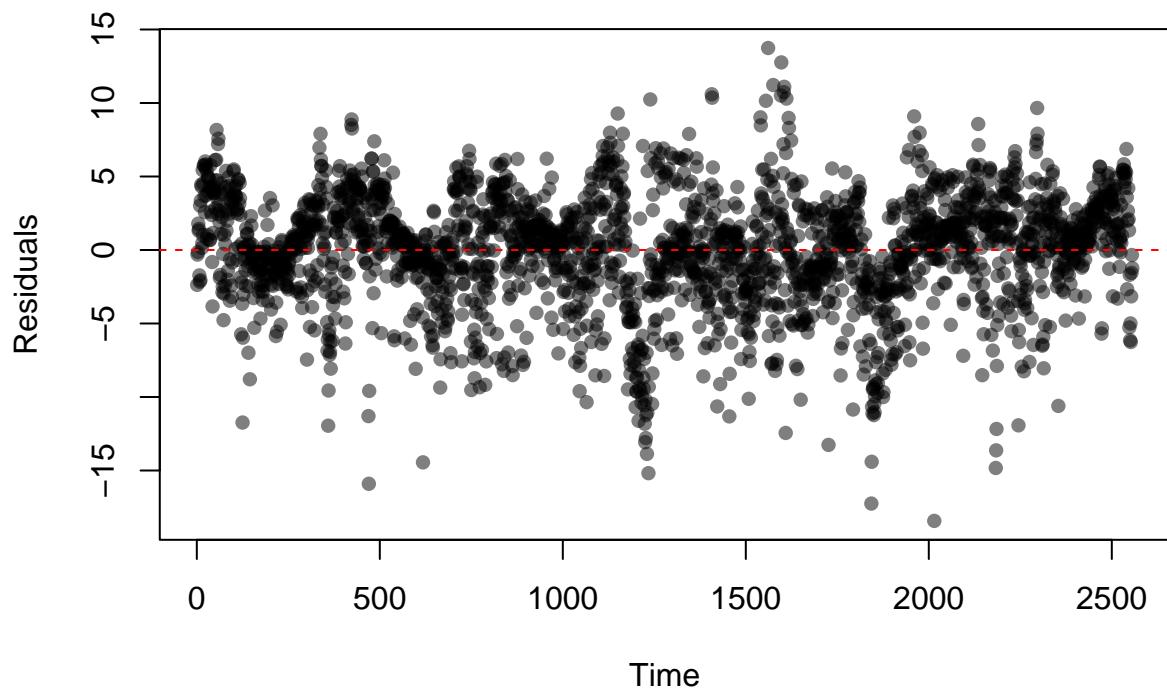
**Residuals  
Normal Q-Q Plot**



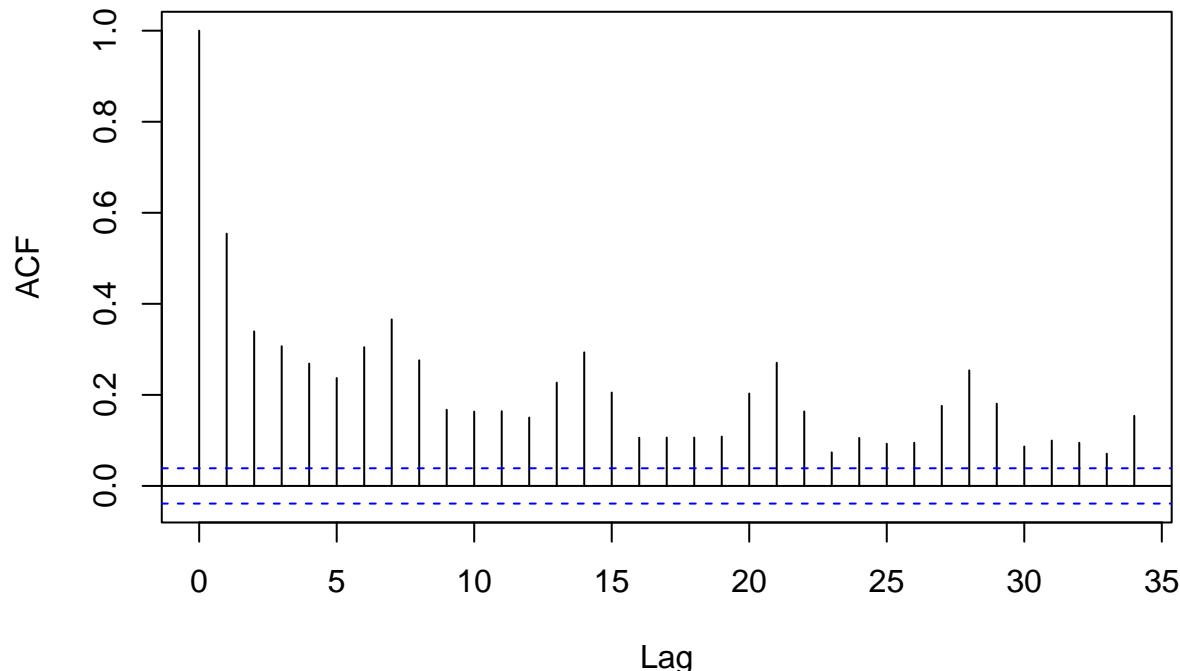
**Fitted vs Residuals**



**Fitted Values  
Residuals vs Time**



## ACF of Residuals



```
# Generate Formal Tests
RegressionDiagnosticsTests(final_reg)

## [1] "--- Shapiro-Wilk Test (Normality) ---"
##
## Shapiro-Wilk normality test
##
## data: my_residuals
## W = 0.97393, p-value < 2.2e-16
##
## [1] "--- Kolmogorov-Smirnov Test (Normality) ---"
##
## Asymptotic one-sample Kolmogorov-Smirnov test
##
## data: scale(my_residuals)
## D = 0.064303, p-value = 1.322e-09
## alternative hypothesis: two-sided
##
## [1] "--- Fligner-Killeen Test (Homoscedasticity) ---"
##
## Fligner-Killeen test of homogeneity of variances
##
## data: my_residuals and seg
## Fligner-Killeen:med chi-squared = 25.293, df = 5, p-value = 0.0001223
##
## [1] "--- Runs Test (Randomness) ---"
##
## Runs Test
##
```

```
## data: my_residuals
## statistic = -21.683, runs = 731, n1 = 1278, n2 = 1278, n = 2556,
## p-value < 2.2e-16
## alternative hypothesis: nonrandomness
```

Diagnostic tests confirm that while the regression model explains a large portion of the variance (Adjusted R<sup>2</sup>=0.80), the residuals significantly violate assumptions of normality, constant variance, and independence. This lack of independence specifically motivates the use of ARIMA modeling in the next phase to capture the remaining autocorrelation structure.