Step 1: Loading in the Data and Initial Plot

```r
# --- 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 error
```

```r
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
```

```r
library(lubridate)
library(MASS)
```

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

```r
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
```

```r
library(randtests)
```

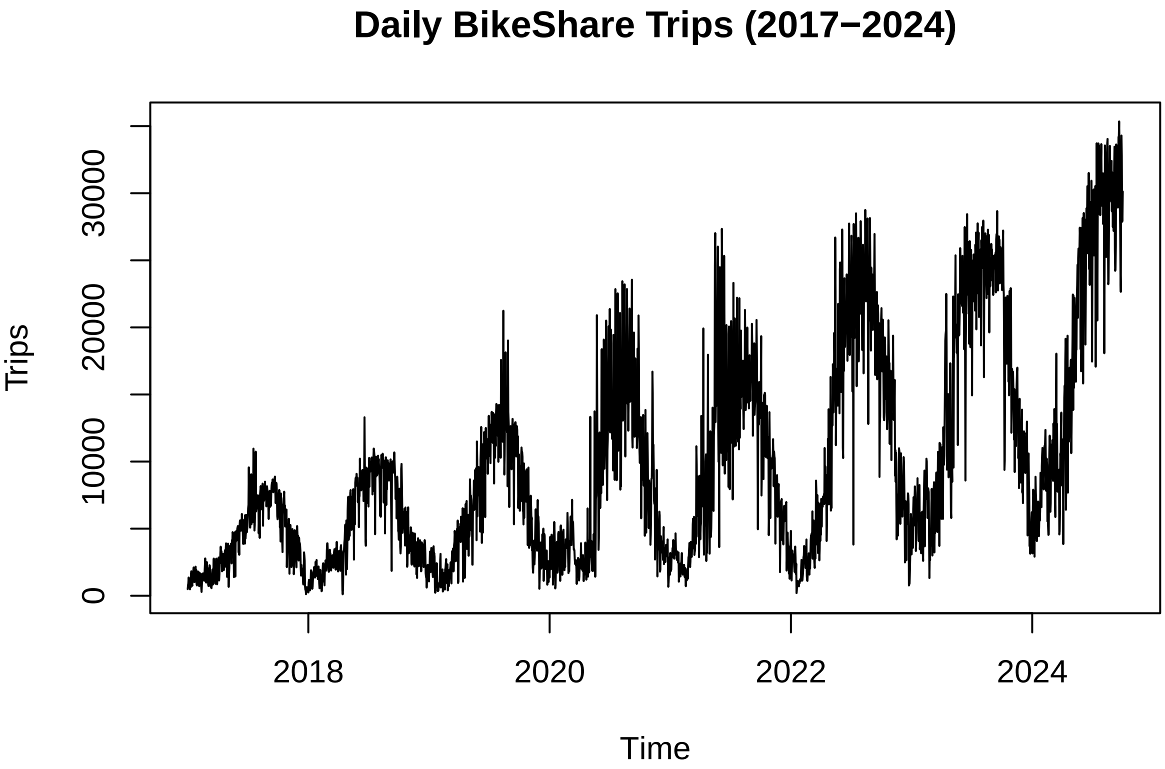
```
## Warning: package 'randtests' was built under R version 4.3.3
```

```
library(forecast) # Needed for ARIMA/HW
```

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

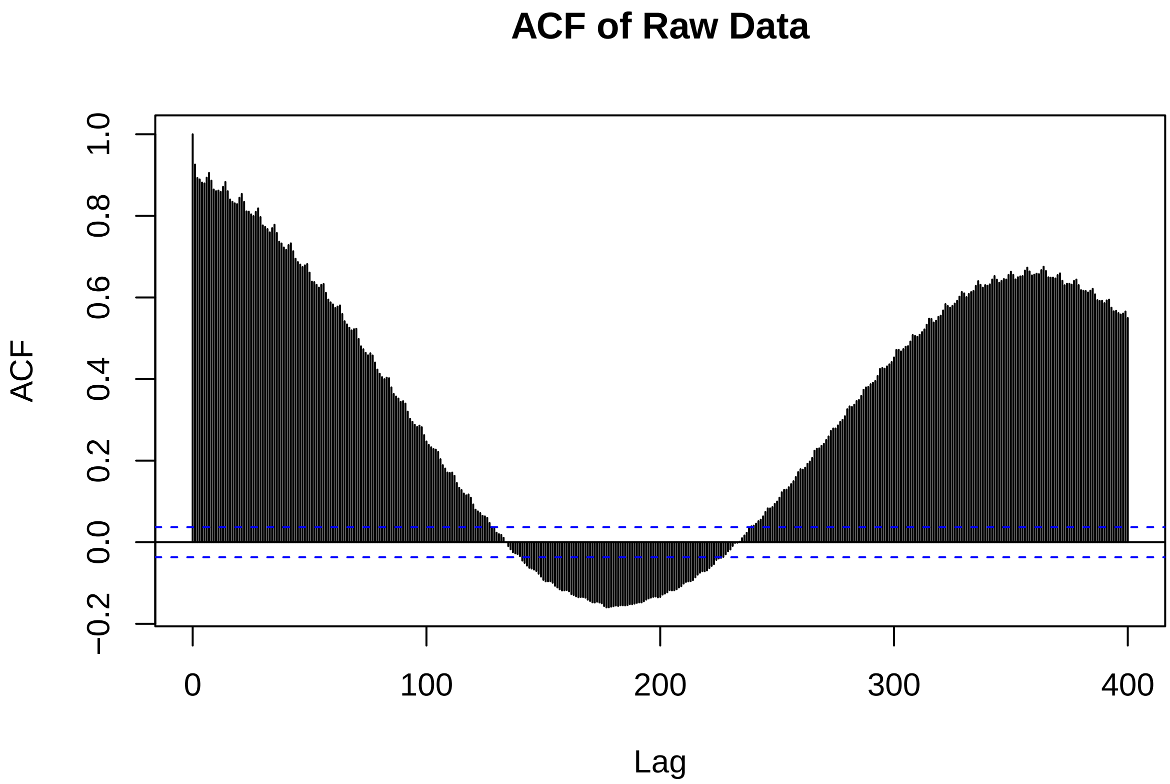
```
## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo
```

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

# Filter for 2017-2024 (Project Scope)
bike <- bike[bike$trip_date >= as.Date("2017-01-01"), ]

# Create Time Series Object (Daily Frequency)
y <- bike$n_trips
bike_ts <- ts(y, start = c(2017, 1), frequency = 365)

# Initial Plot
plot(bike_ts, main = "Daily BikeShare Trips (2017-2024)", ylab = "Trips")
```

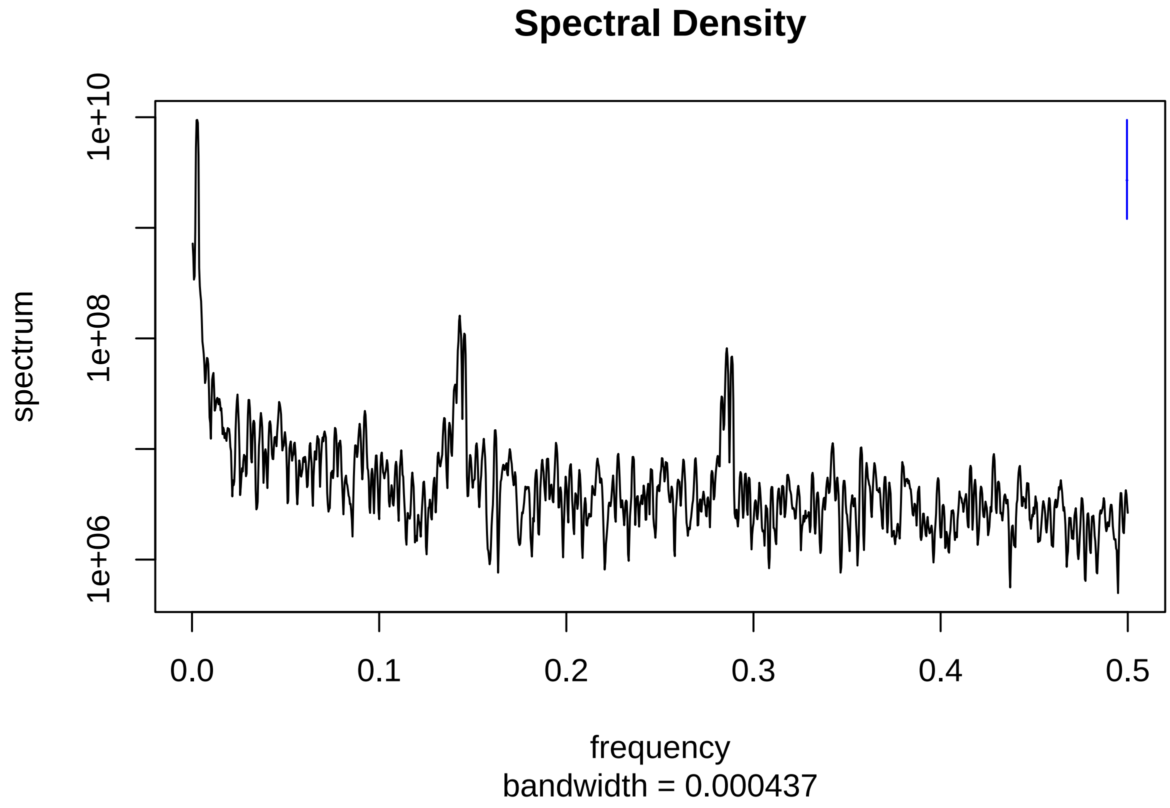


Step 2: Checking ACF and Period

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



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

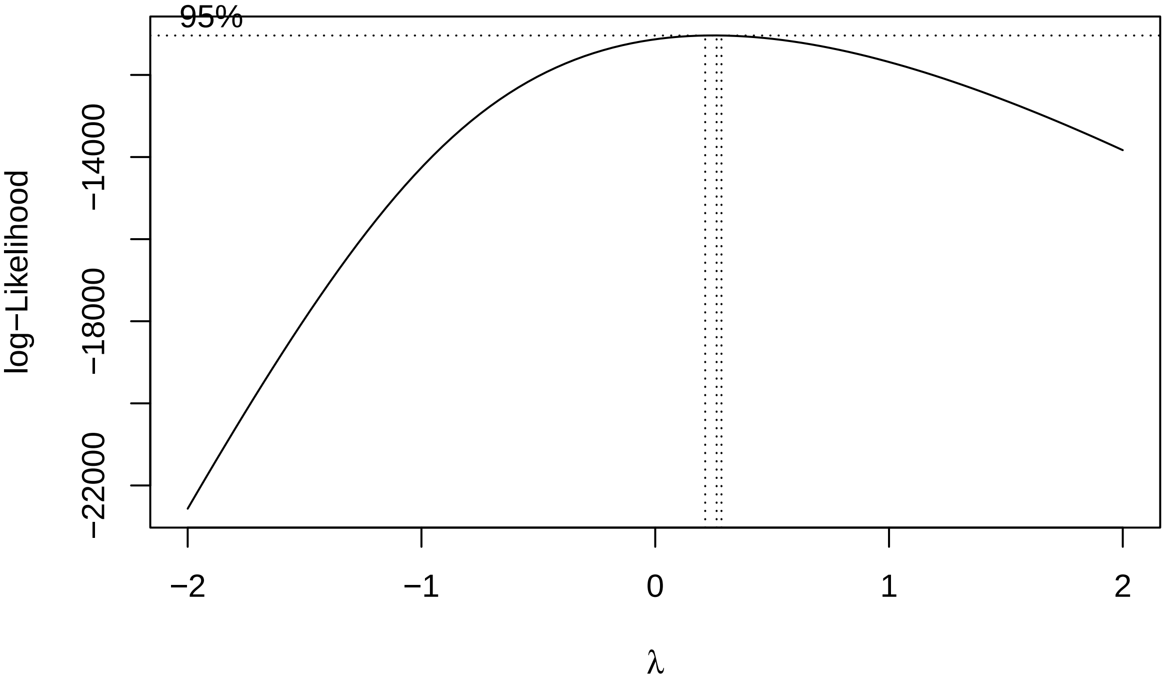


```
# Check dominant period
print(1 / spec$freq[which.max(spec$spec)])
```

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

Step 3: Find Lambda and Do BoxCox Transformation

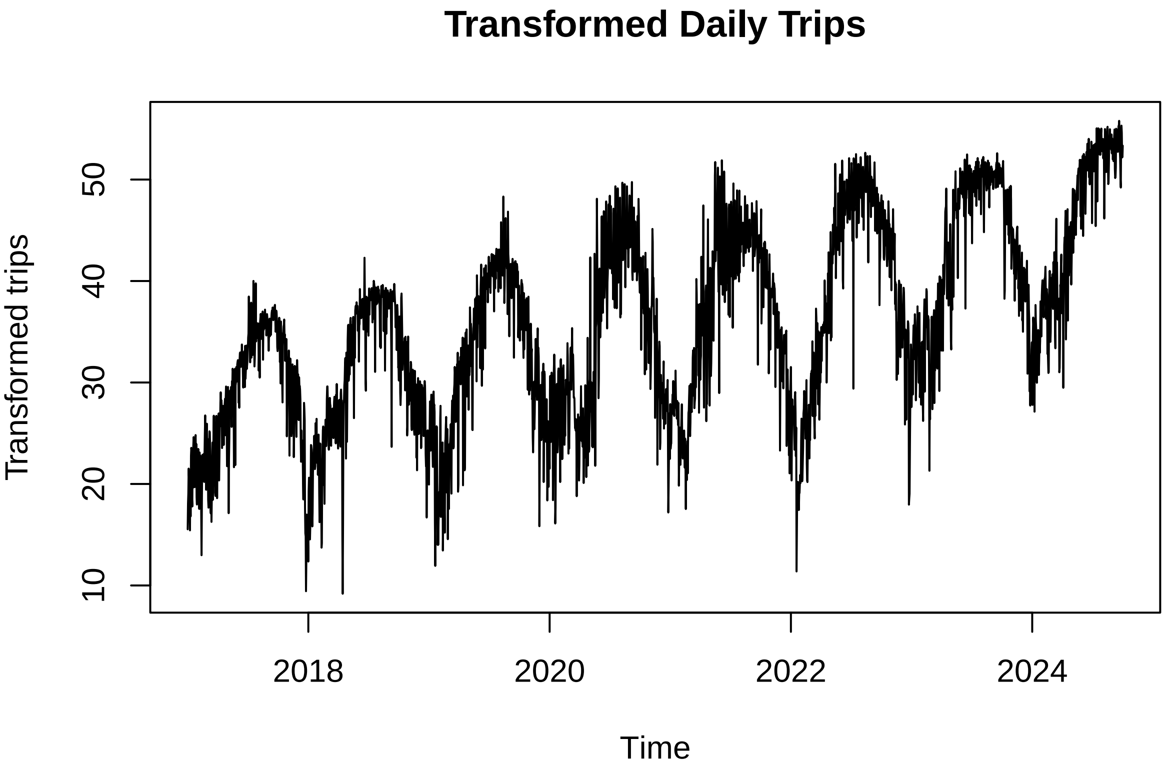
```
# Box-Cox Check
bc <- boxcox(lm(y ~ 1), lambda = seq(-2, 2, 0.1))
```



```
lam <- bc$x[which.max(bc$y)]
print(paste("Optimal Lambda:", lam))
```

```
## [1] "Optimal Lambda: 0.262626262626263"
```

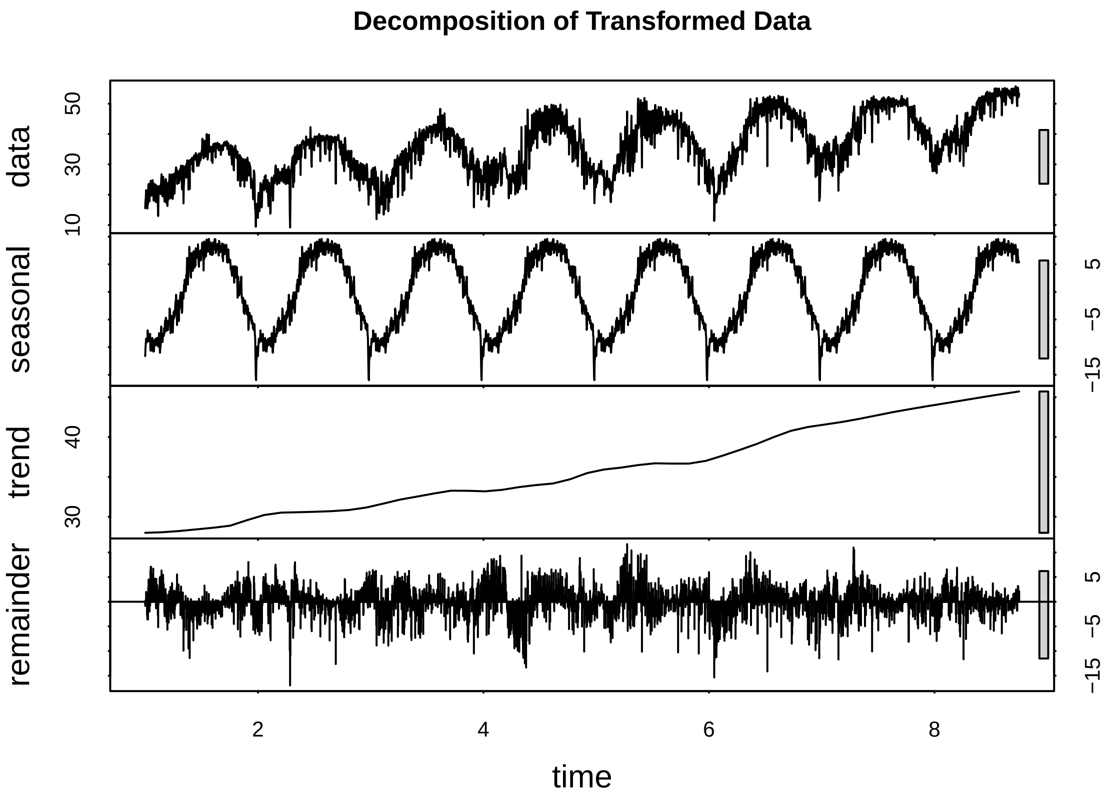
```
if (lam == 0) {
  y_trans <- log(bike_ts)
} else if (lam > 0) {
  y_trans <- (bike_ts^lam - 1) / lam
} else {
  # negative lambda → use minus sign trick like in lectures
  y_trans <- -(bike_ts^lam)
}

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



```
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")
```



Step 5: Mutating the Bike Object to add Seasonality Factors

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

    # 2. Factor-based Seasonality
    month_fac = as.factor(month(trip_date)),
    weekday_fac = as.factor(wday(trip_date, label=TRUE)),
    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),
    cos_year = cos(2 * pi * time_index / 360)
  )

# 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

```r
# --- Model Fitting ---

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

# 2. Trend + Fourier + Month + Day of Week
```

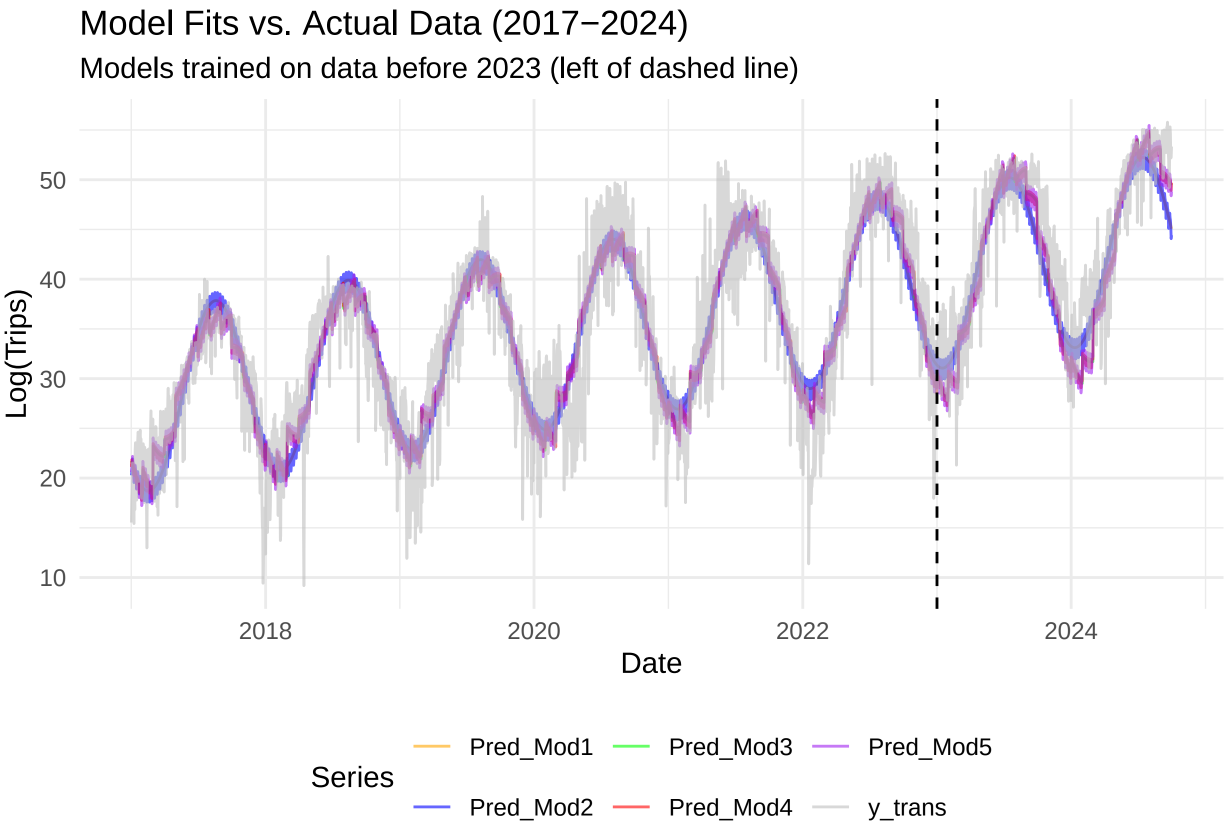
```r
mod2 <- lm(y_trans ~ time_index + sin_year + cos_year + weekday_fac, data = train_seas)

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

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

# 5. Trend + Fourier + Month + Day of Week
mod5 <- lm(y_trans ~ time_index + sin_year + cos_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                    16.51951 12338.62 0.7785712          4
## Trend + Fourier + DayOfWeek        16.03413 12295.23 0.7835148         10
## Trend + Fourier + Month            15.76644 12145.57 0.7982779         15
## Trend + Fourier + Month + Weekend  15.49865 12125.55 0.8002066         16
## Trend + Fourier + Month + DayOfWeek 15.32634 12097.48 0.8032123         21
```

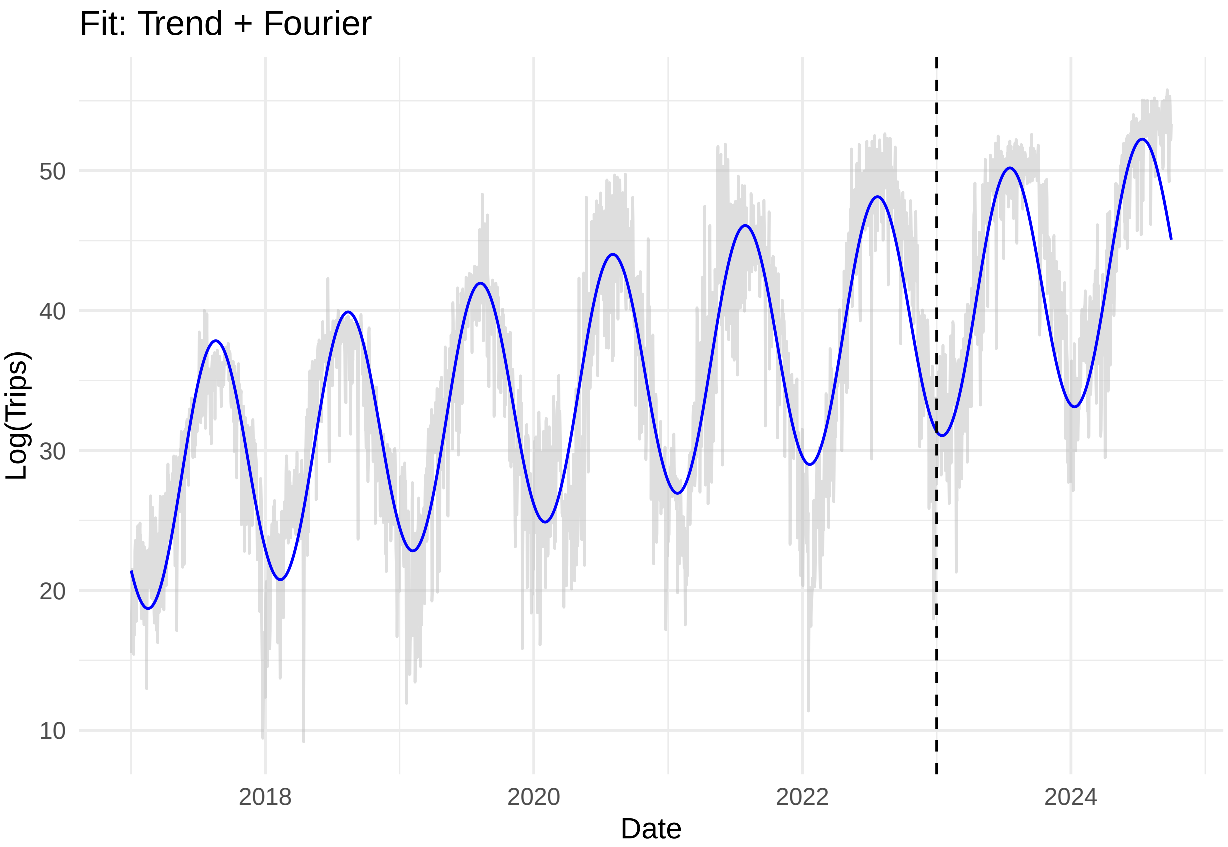
```
# --- Visualizing Fits on Full Data (2017-Present) ---

# 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, cos_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 = "Log(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")
```



```r
# 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 = "Log(Trips)", x = "Date") +
    theme_minimal()
}
```

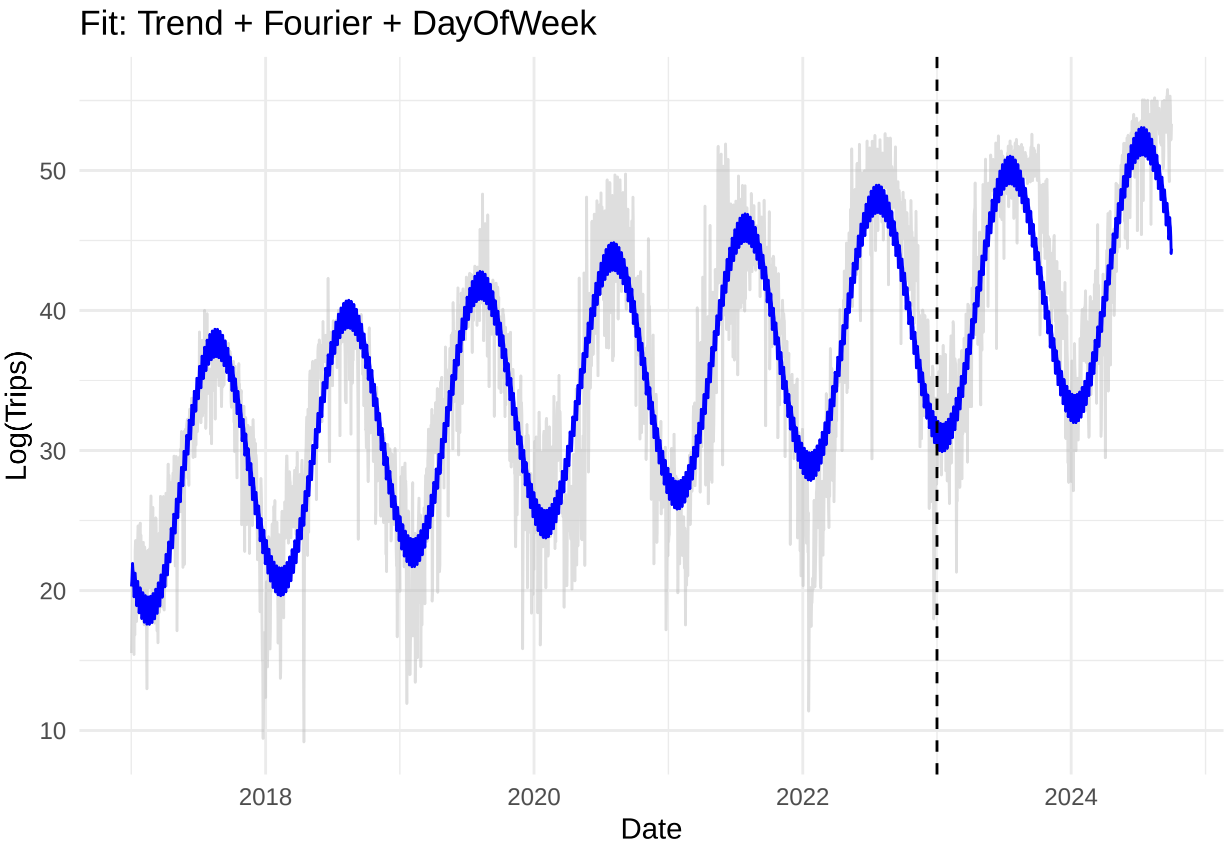
```r
plot_one_model("Pred_Mod1", "Trend + Fourier")
```

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



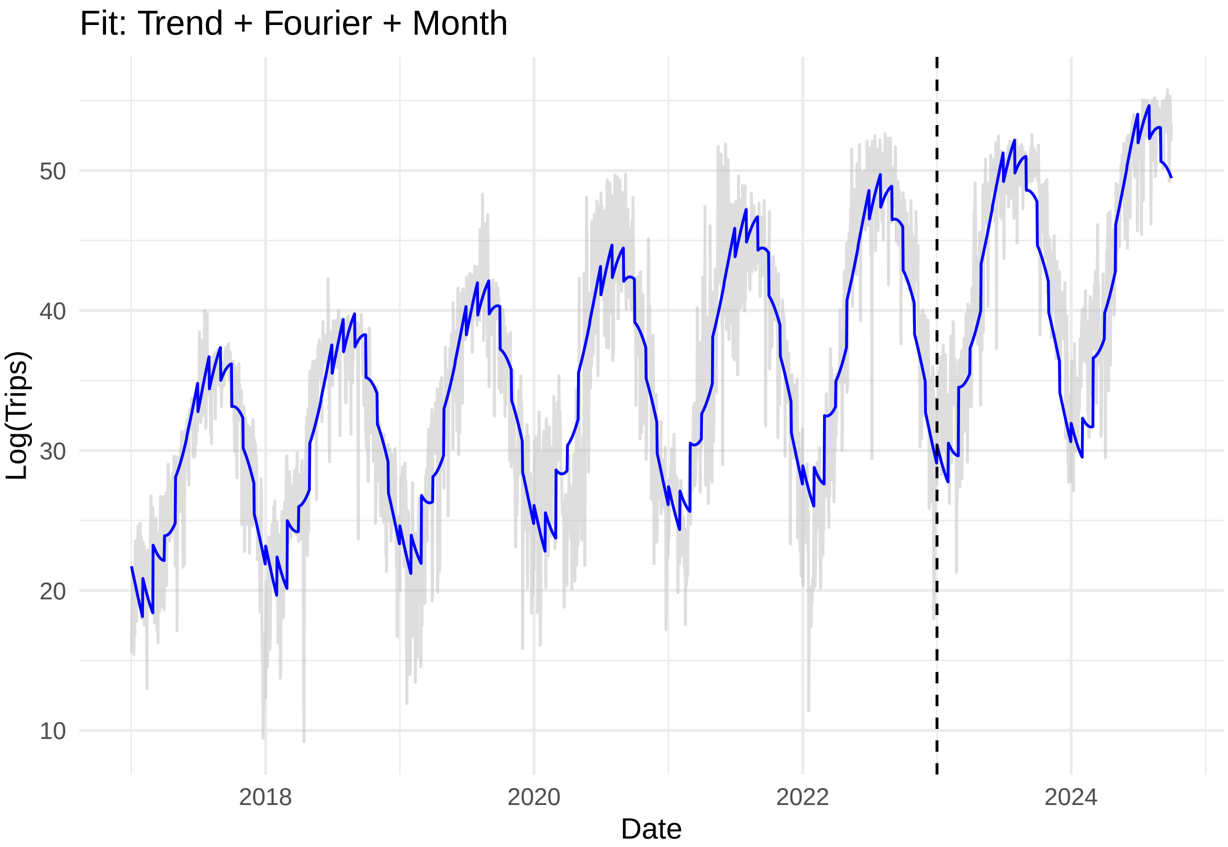
```
plot_one_model("Pred_Mod2", "Trend + Fourier + DayOfWeek")
```

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



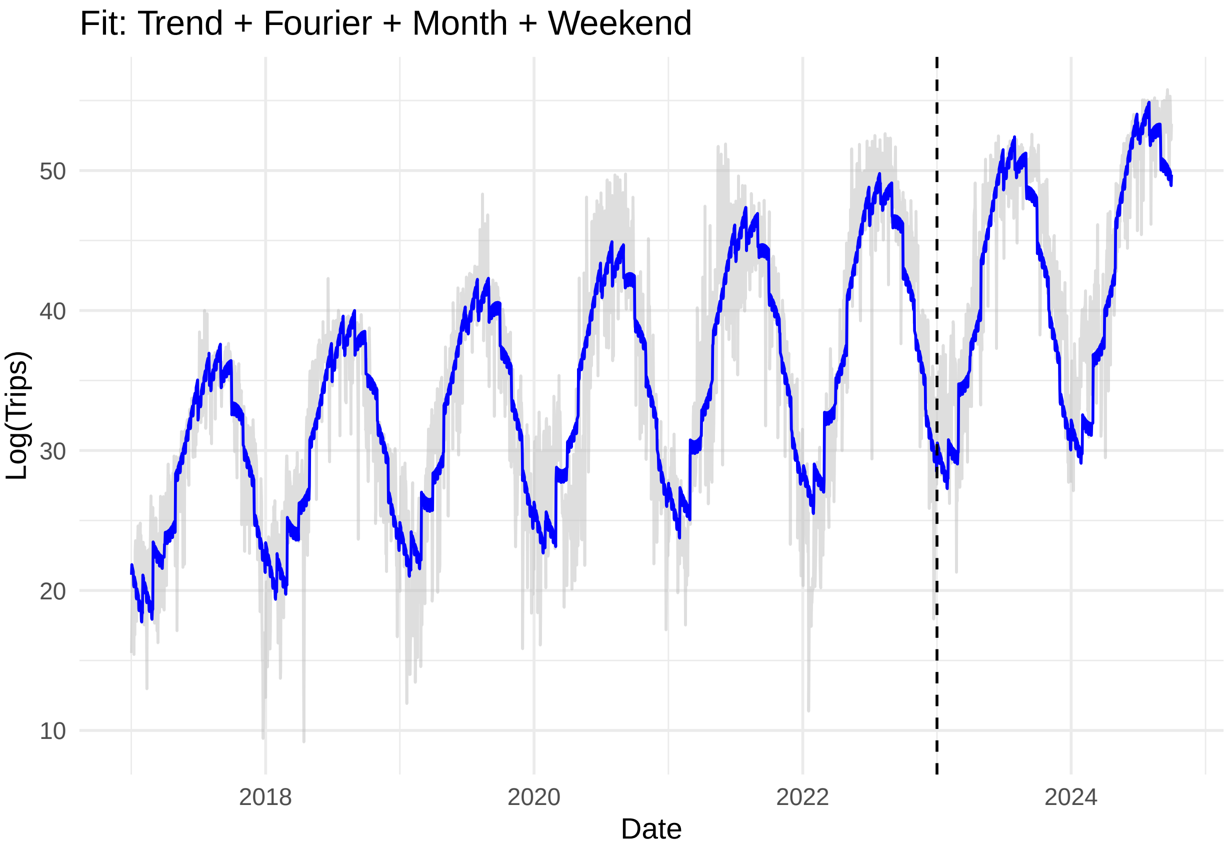
```r
plot_one_model("Pred_Mod3", "Trend + Fourier + Month")
```

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



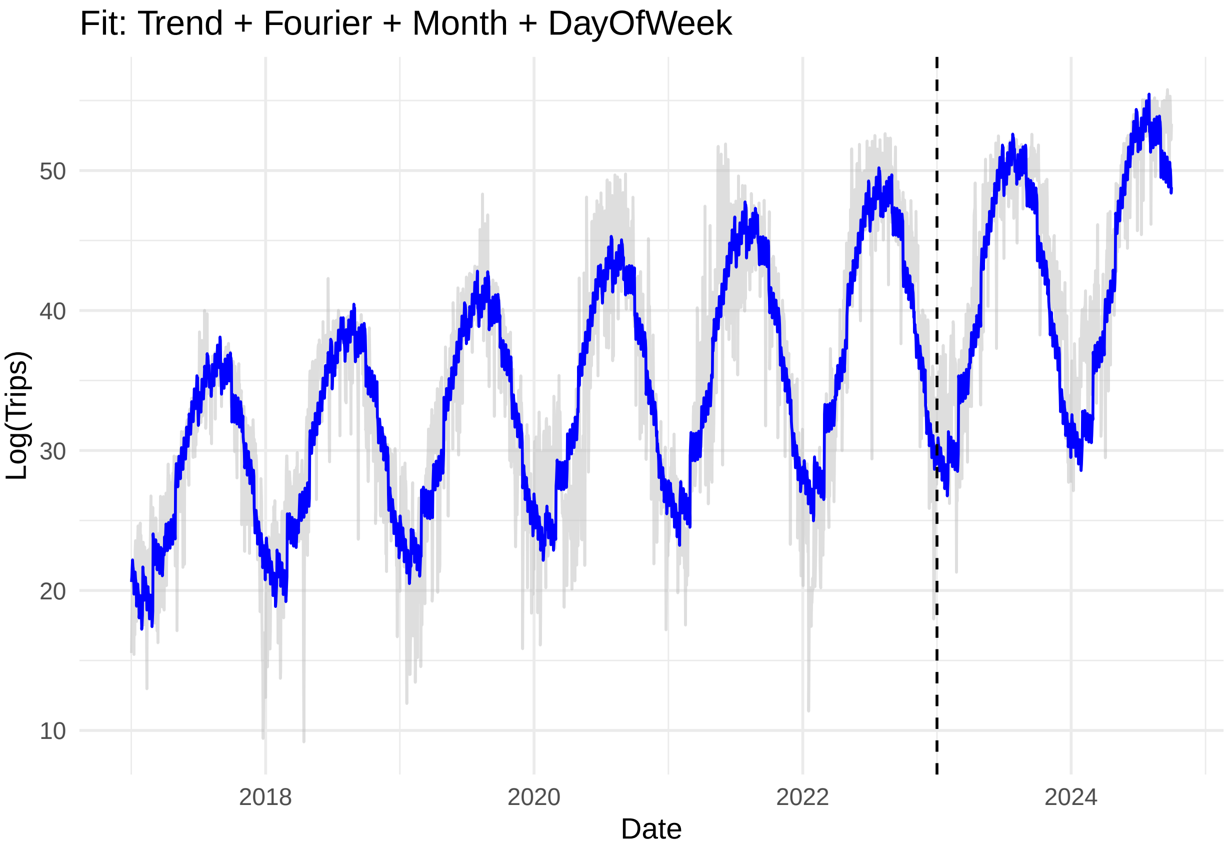
```r
plot_one_model("Pred_Mod4", "Trend + Fourier + Month + Weekend")
```

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



```
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.
```



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..."
```

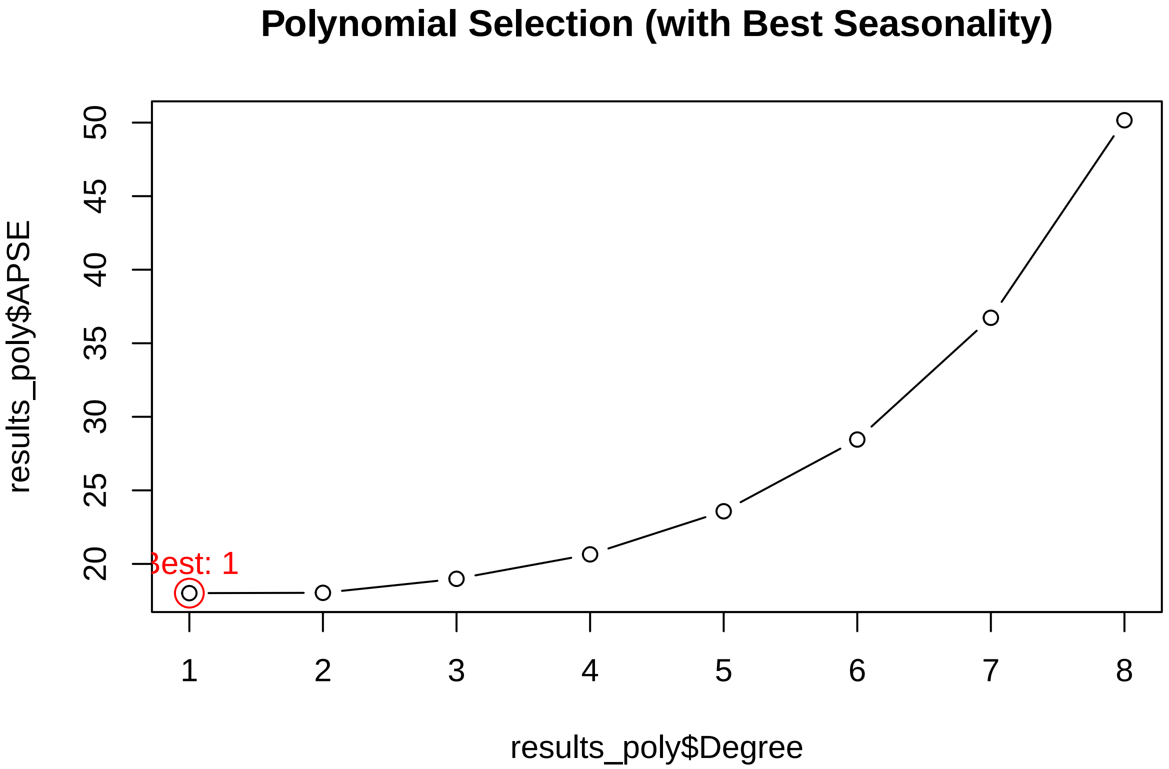
```
# Limit CV to pre-2024 data to avoid peeking at the final test set
cv_data <- subset(bike, trip_date < "2024-01-01")
n_cv <- nrow(cv_data)

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)
```

```r
  # 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],
      cos_year = cv_data$cos_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$sin_year[test_idx],
      cos_year = cv_data$cos_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: 18.0131"
## [1] "Degree 2 APSE: 18.0349"
## [1] "Degree 3 APSE: 18.9866"
## [1] "Degree 4 APSE: 20.6492"
## [1] "Degree 5 APSE: 23.5771"
## [1] "Degree 6 APSE: 28.4535"
## [1] "Degree 7 APSE: 36.7309"
## [1] "Degree 8 APSE: 50.1599"
```

```r
# Plot Results
plot(results_poly$Degree, results_poly$APSE, type='b', main="Polynomial Selection (with Best Seasonalit
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")
```



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

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

As anticipated, the best model is a linear polynomial.

Step 7: Residual Diagnostics on the Selected Model

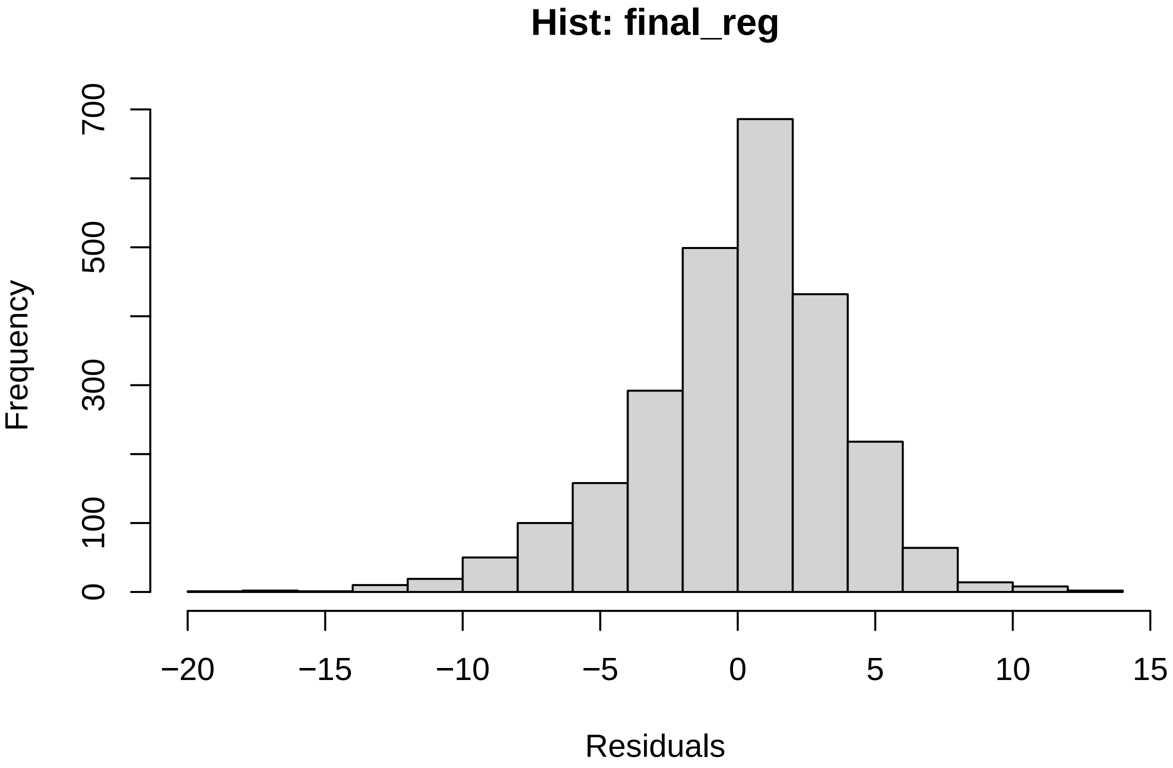
```
# --- Fit Final Regression Model (REQUIRED before Diagnostics) ---

# 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,
  cos_year = train_full$cos_year
)
```

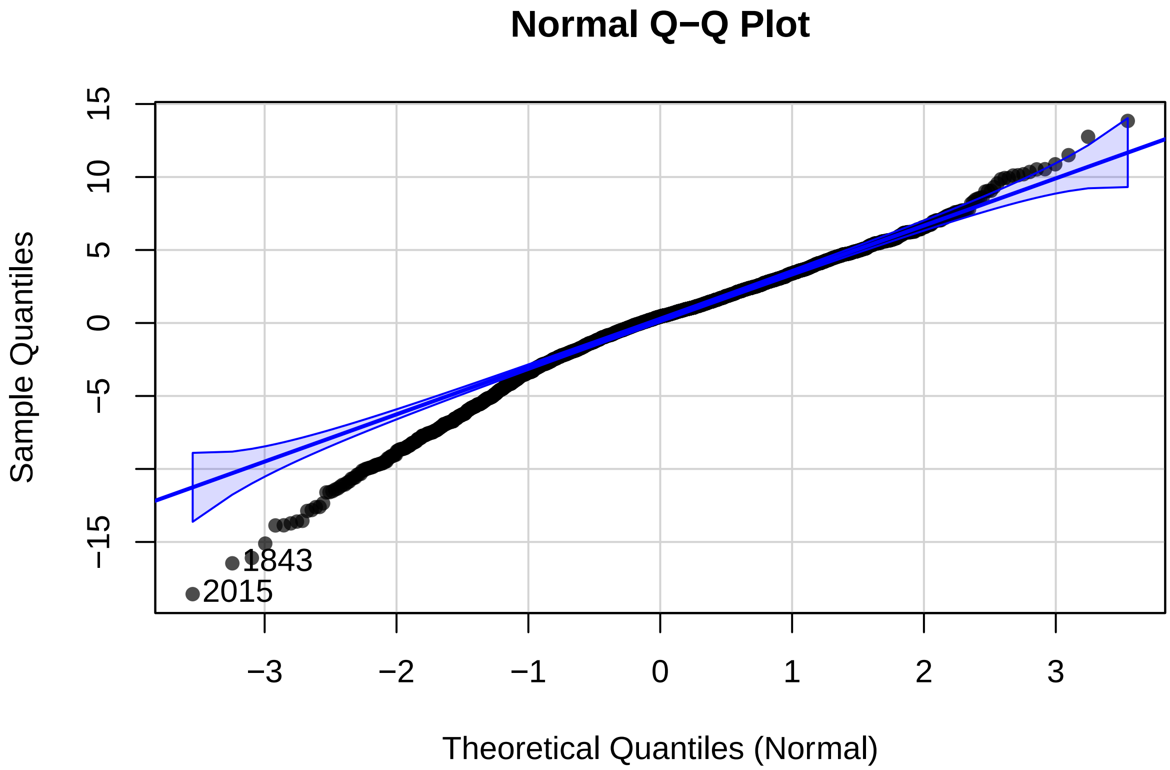
```
# 4. Fit the model
# This creates the 'final_reg' object the next chunk is looking for
final_reg <- lm(y ~ ., data = final_train_data)

# Check summary to confirm it worked
summary(final_reg)
```

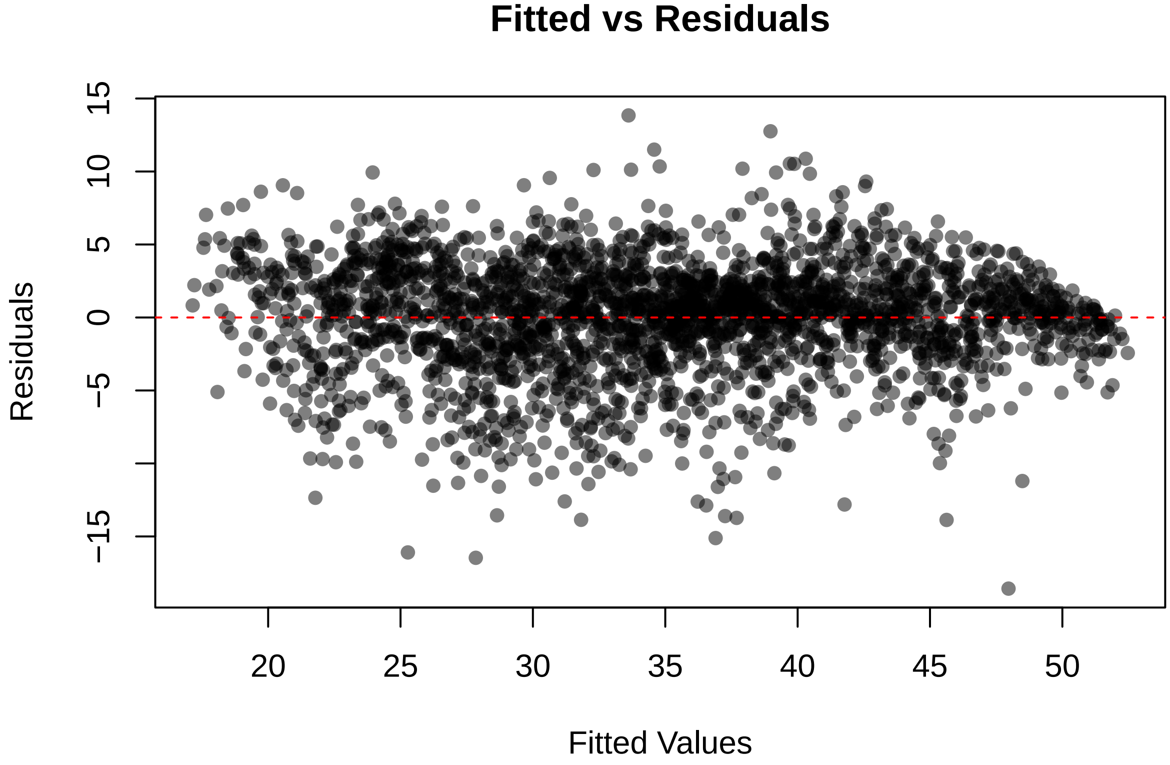
```
##
## Call:
## lm(formula = y ~ ., data = final_train_data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -18.5736  -1.9741   0.4343   2.3876  13.8414
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) 20.9318752  0.5204347  40.220  < 2e-16 ***
## X1           0.0062848  0.0001022  61.515  < 2e-16 ***
## month2       2.0486230  0.4325809   4.736 2.30e-06 ***
## month3       6.1068442  0.5627610  10.852  < 2e-16 ***
## month4       8.2183982  0.7177195  11.451  < 2e-16 ***
## month5      11.5946221  0.8431696  13.751  < 2e-16 ***
## month6      12.0622577  0.9259733  13.027  < 2e-16 ***
## month7      10.6274905  0.9498652  11.188  < 2e-16 ***
## month8       8.9284136  0.9159362   9.748  < 2e-16 ***
## month9       7.2412258  0.8300829   8.723  < 2e-16 ***
## month10      4.3078182  0.6989622   6.163 8.27e-10 ***
## month11      1.7988811  0.5495514   3.273  0.00108 **
## month12     -0.9631780  0.4123201  -2.336  0.01957 *
## weekday.L    1.0166644  0.1973698   5.151 2.79e-07 ***
## weekday.Q   -1.3007301  0.1973887  -6.590 5.34e-11 ***
## weekday.C   -0.1413682  0.1974011  -0.716  0.47397
## weekday^4    0.2880981  0.1974190   1.459  0.14460
## weekday^5   -0.1722256  0.1974342  -0.872  0.38312
## weekday^6   -0.1438771  0.1974375  -0.729  0.46624
## sin_year    -6.0075095  0.4458561 -13.474  < 2e-16 ***
## cos_year     0.1052805  0.4520442   0.233  0.81586
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.772 on 2535 degrees of freedom
## Multiple R-squared:  0.8258, Adjusted R-squared:  0.8244
## F-statistic: 600.7 on 20 and 2535 DF,  p-value: < 2.2e-16
```

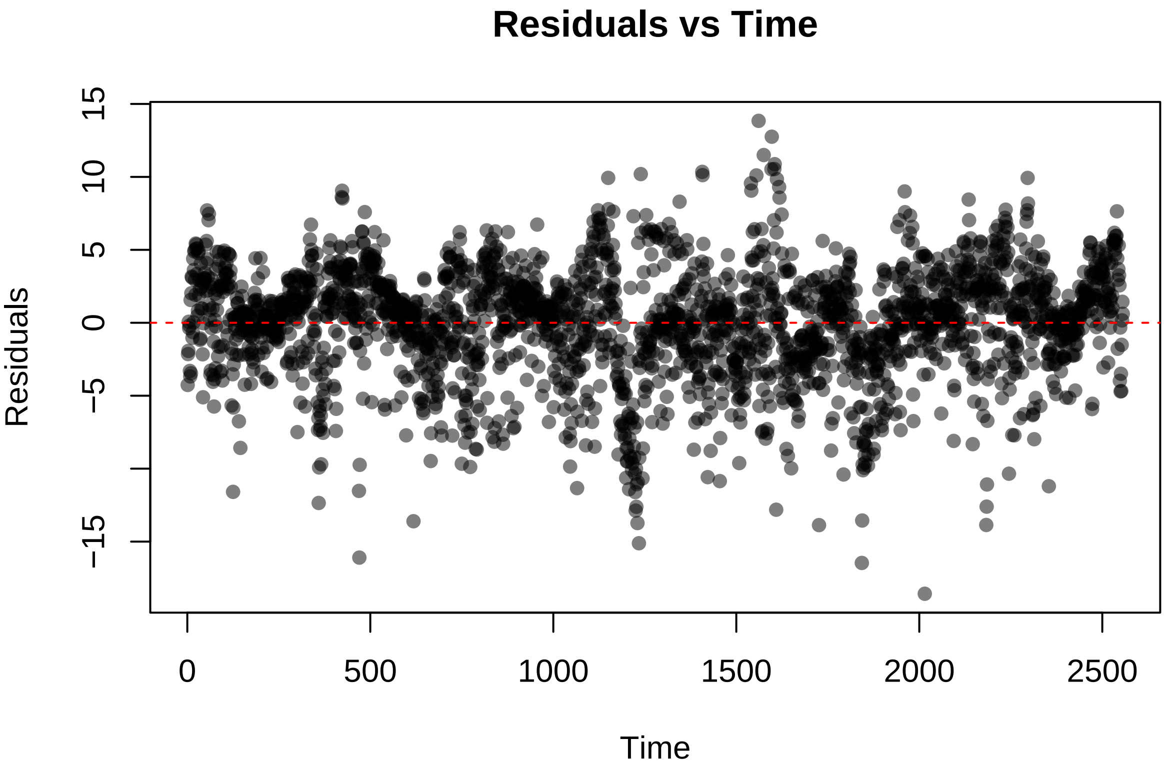
```
# --- 1. Define Class Diagnostic Functions ---

RegressionDiagnosicsPlots <- 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))
```

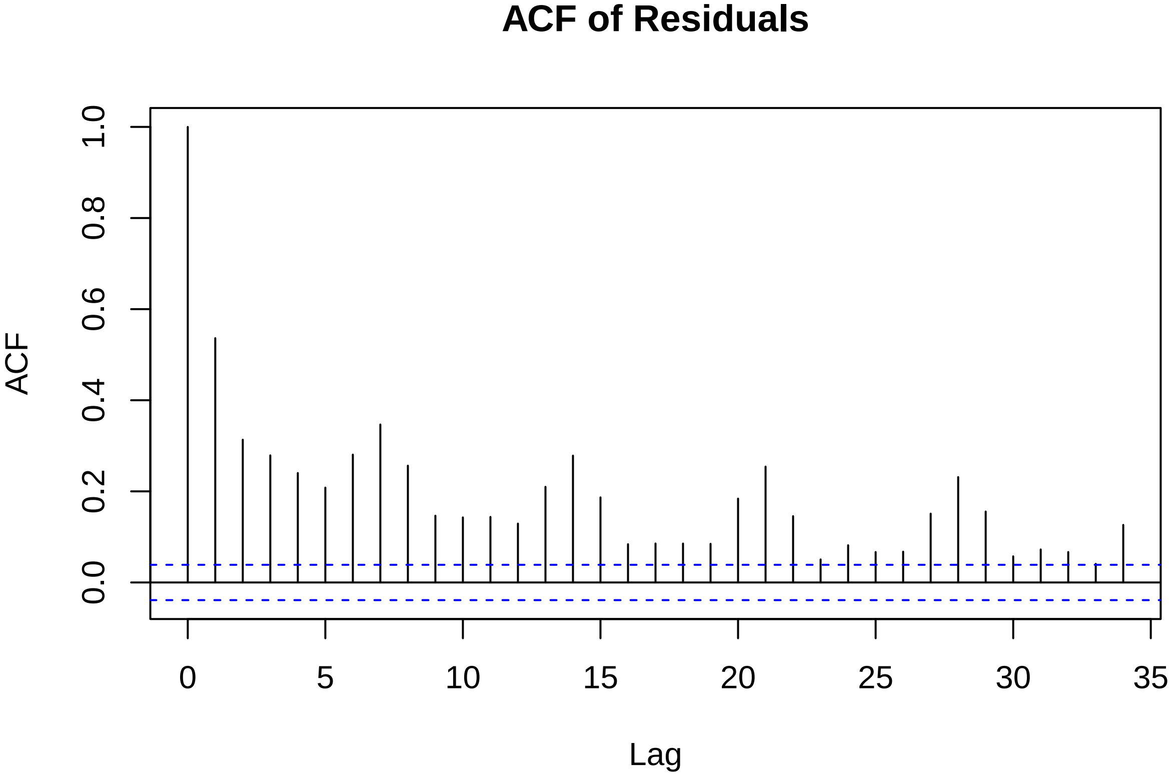
```r
  } else {
    model_name <- title_suffix
  }

  # 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
}

RegressionDiagnosicsTests <- 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
RegressionDiagnosicsPlots(final_reg)
```




Normal Q−Q Plot


Fitted vs Residuals





```r
# Generate Formal Tests
RegressionDiagnosicsTests(final_reg)
```

```
## [1] "--- Shapiro-Wilk Test (Normality) ---"
##
##  Shapiro-Wilk normality test
```

```
##
## data:  my_residuals
## W = 0.97523, p-value < 2.2e-16
##
## [1] "--- Kolmogorov-Smirnov Test (Normality) ---"
##
##  Asymptotic one-sample Kolmogorov-Smirnov test
##
## data:  scale(my_residuals)
## D = 0.065566, p-value = 5.713e-10
## 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 = 36.469, df = 5, p-value = 7.652e-07
##
## [1] "--- Runs Test (Randomness) ---"
##
##  Runs Test
##
## data:  my_residuals
## statistic = -21.248, runs = 742, 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 R2=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.