Step 1: Loading in the Data

```r
# Packages used in tutorials
library(MASS)      # boxcox
library(car)       # qqPlot
library(randtests) # runs.test
# library(forecast) # OPTIONAL if you want auto.arima, not required

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

bike$trip_date <- as.Date(bike$trip_date)

# Filter the bike dataframe to keep only trips from 2017 onwards
bike <- bike[bike$trip_date >= as.Date("2017-01-01"), ]

# Verify the new range
range(bike$trip_date)
```

```
## [1] "2017-01-01" "2024-09-30"
```

```r
str(bike)
```

```
## 'data.frame':    2830 obs. of  2 variables:
##  $ trip_date: Date, format: "2017-01-01" "2017-01-02" ...
##  $ n_trips  : int  487 775 918 1359 1202 1189 632 477 1269 622 ...
```

```r
head(bike)
```

```
##    trip_date n_trips
## 1 2017-01-01     487
## 2 2017-01-02     775
## 3 2017-01-03     918
## 4 2017-01-04    1359
## 5 2017-01-05    1202
## 6 2017-01-06    1189
```

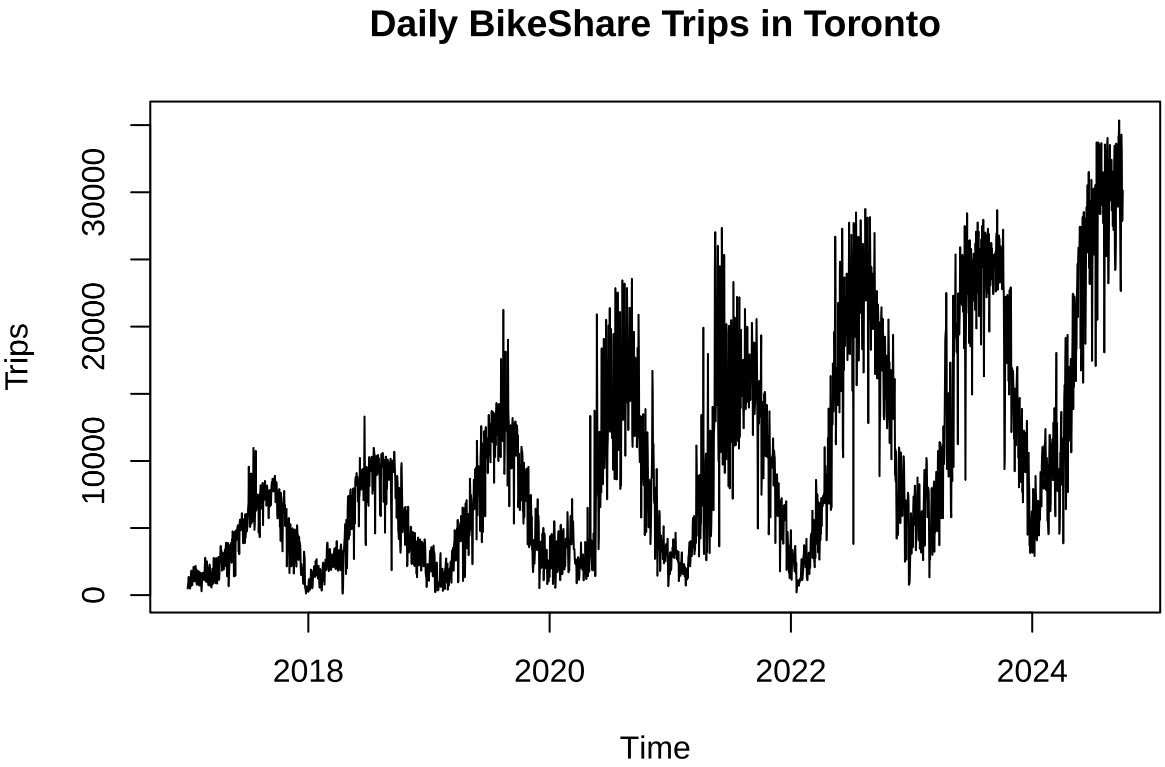
```r
range(bike$trip_date)
```

```
## [1] "2017-01-01" "2024-09-30"
```

Initial Plotting for Time Series

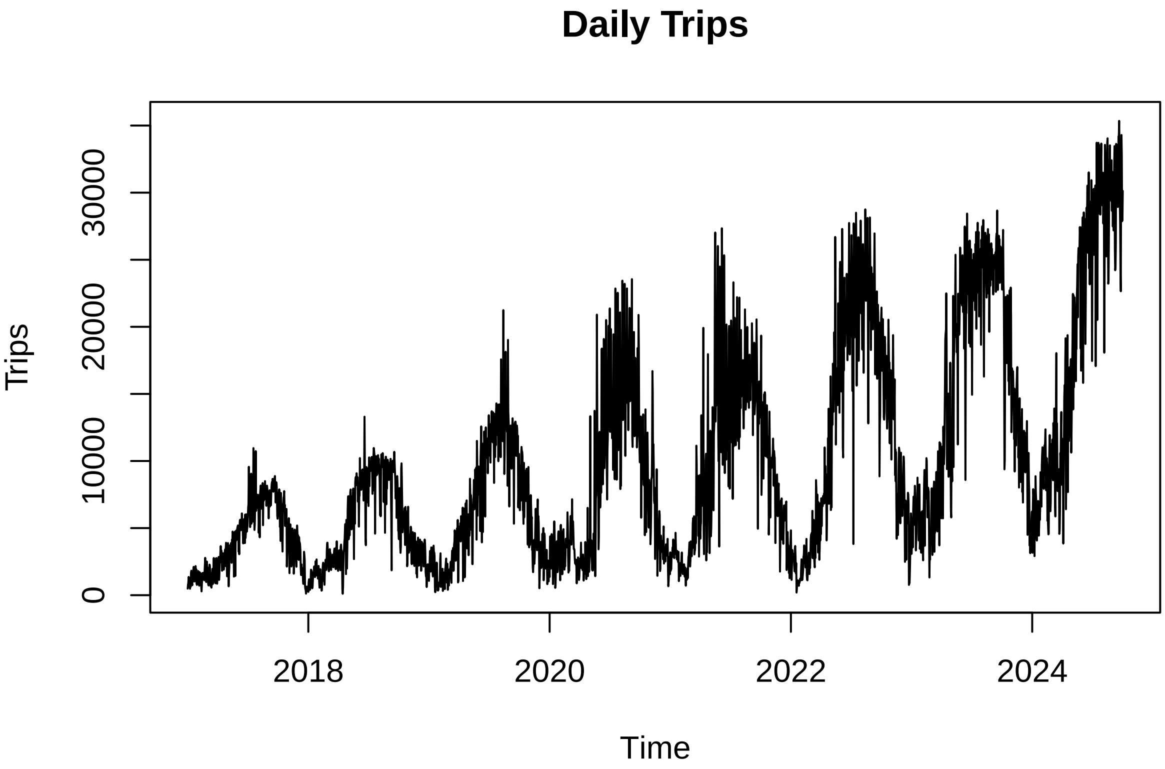
```r
# Sort just in case
bike <- bike[order(bike$trip_date), ]

# Extract response as a vector
y <- bike$n_trips

# Daily frequency with yearly seasonality (approx 365)
bike_ts <- ts(
  y,
  start = c(as.numeric(format(min(bike$trip_date), "%Y")),
            as.numeric(format(min(bike$trip_date), "%j"))),
  frequency = 365
)
```

```r
plot(bike_ts, main = "Daily BikeShare Trips in Toronto", ylab = "Trips")
```

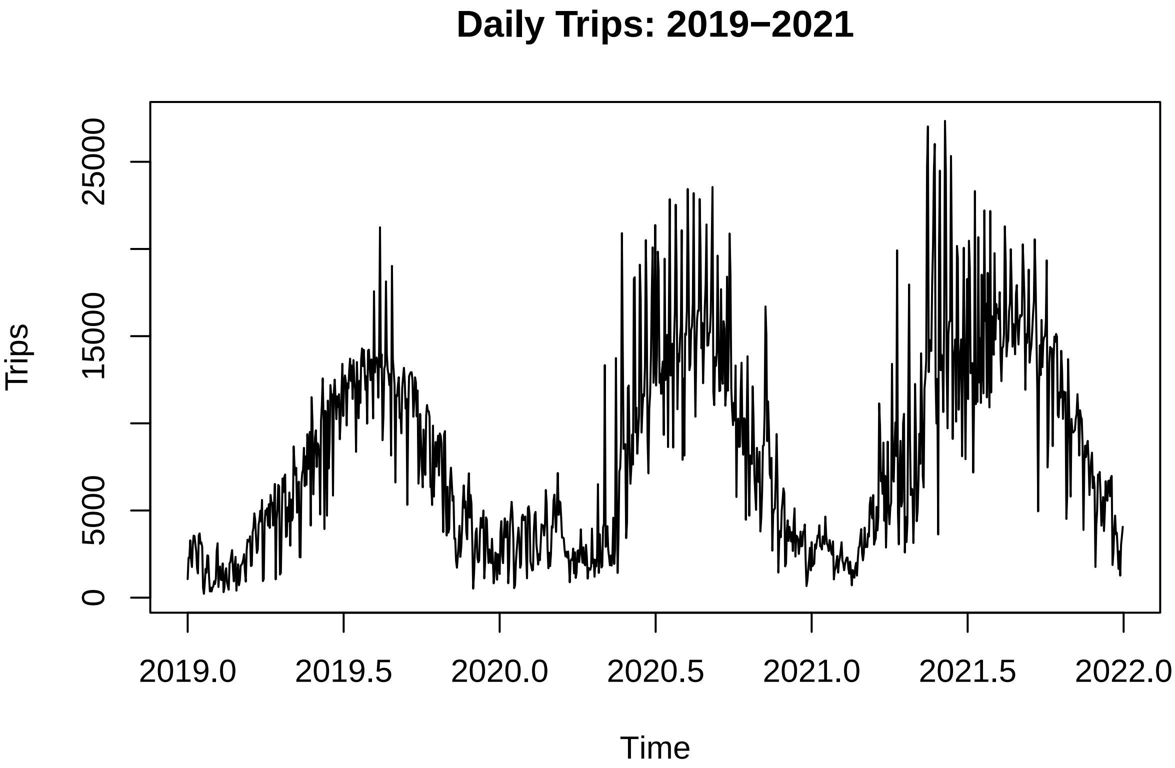


Step 2: EDA

```r
par(mfrow = c(1, 1))
plot(bike_ts, main = "Daily Trips", ylab = "Trips")
```

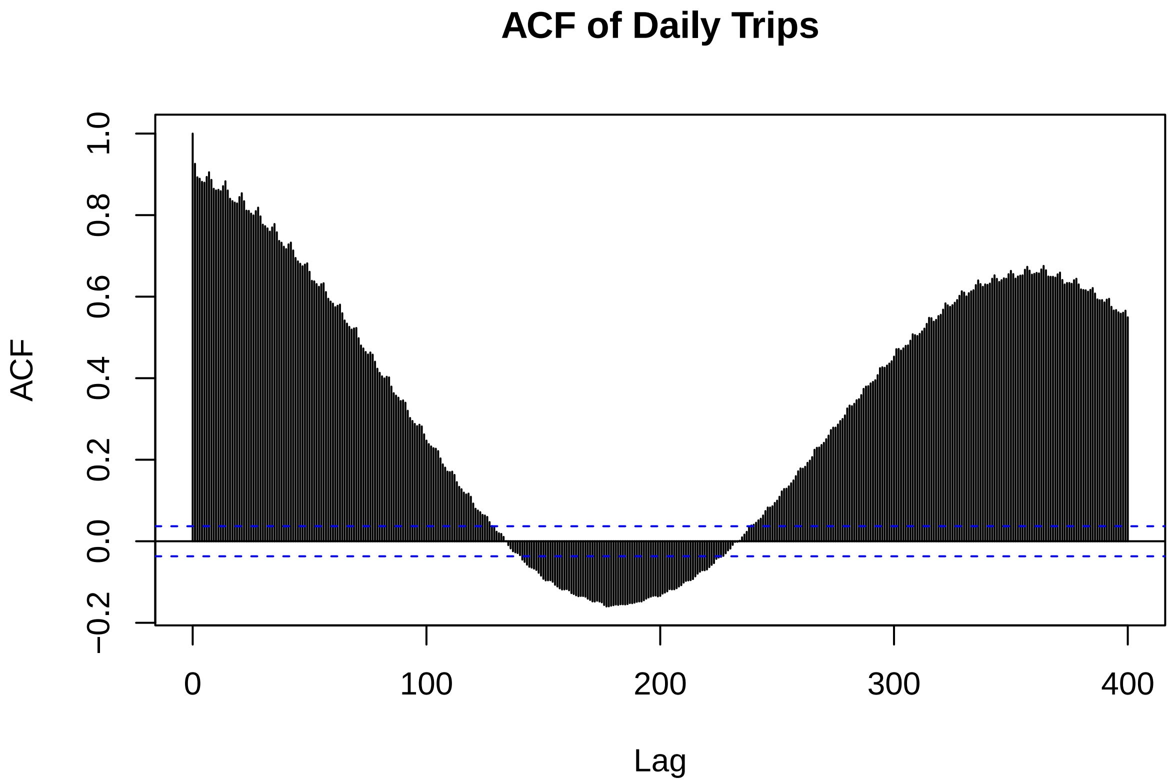


```r
# maybe a zoom on a couple of years
plot(window(bike_ts, start = c(2019, 1), end = c(2021, 365)),
     main = "Daily Trips: 2019-2021", ylab = "Trips")
```

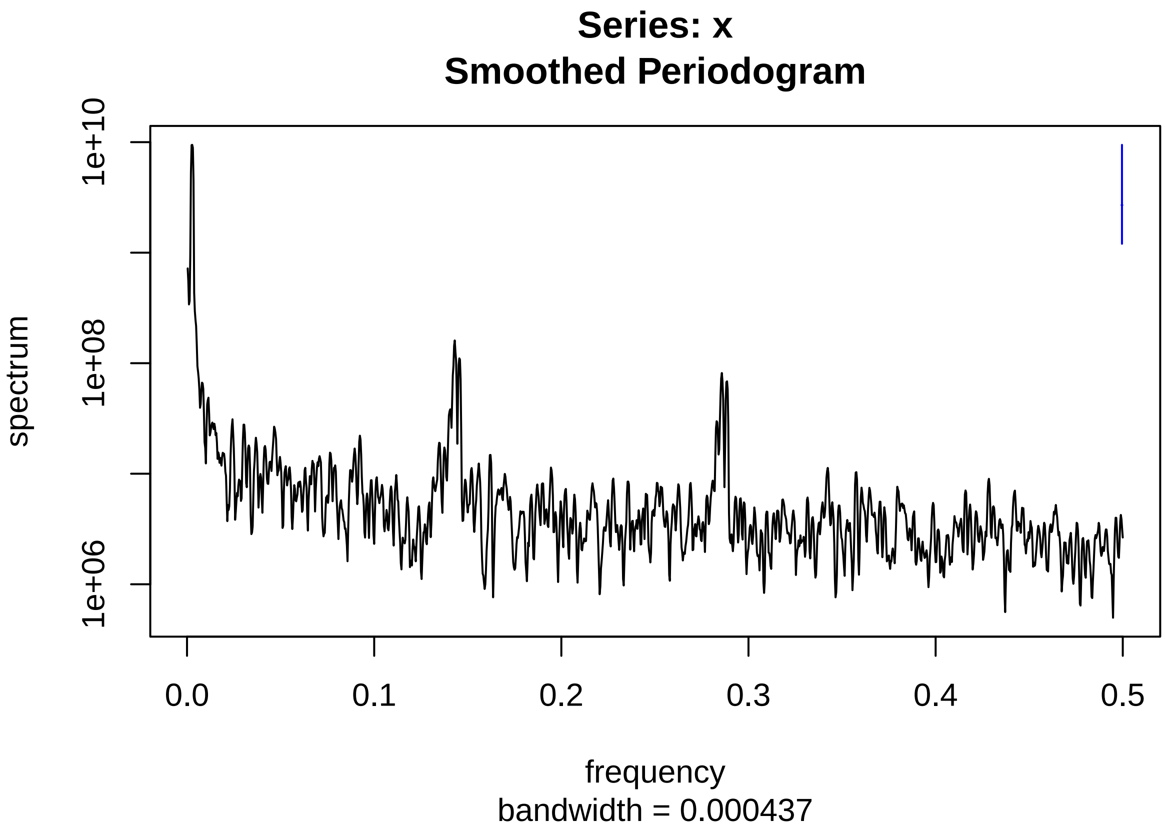


Confirming Seasonality with ACF and Spectrum

```
acf(as.vector(bike_ts), lag.max = 400,
    main = "ACF of Daily Trips")
```



```
spec_bike <- spectrum(as.vector(bike_ts), spans = 5)
```

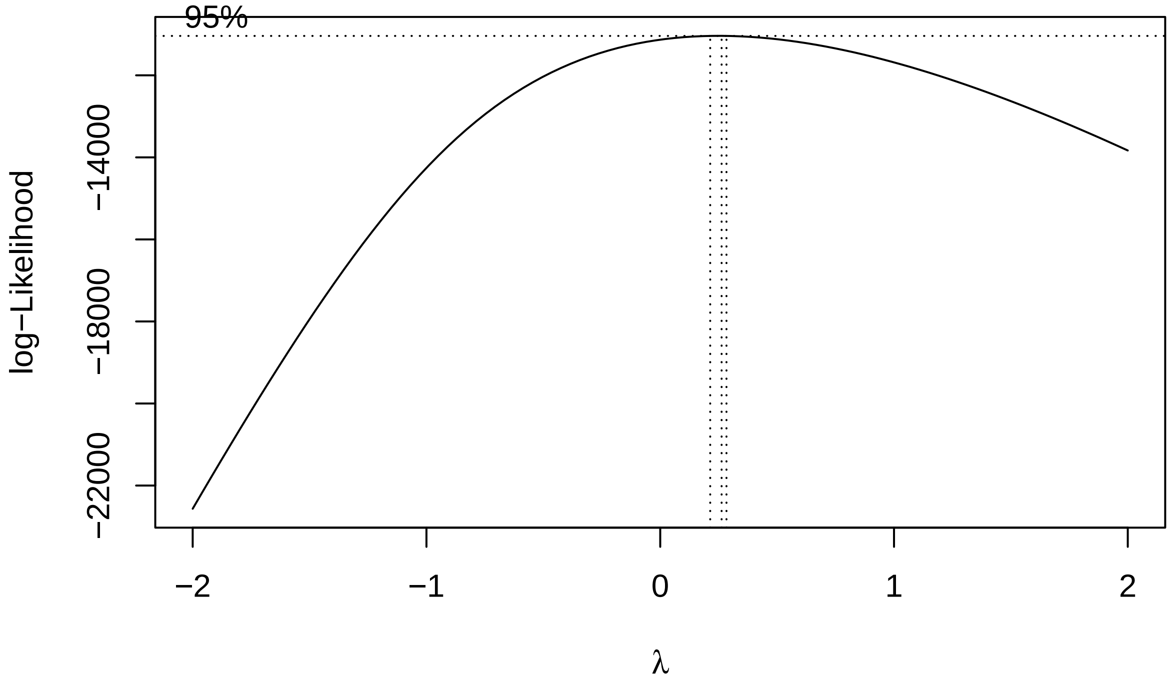


```
1 / spec_bike$freq[which.max(spec_bike$spec)]  # estimated period
```

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

Step 3: Box-Cox transformation

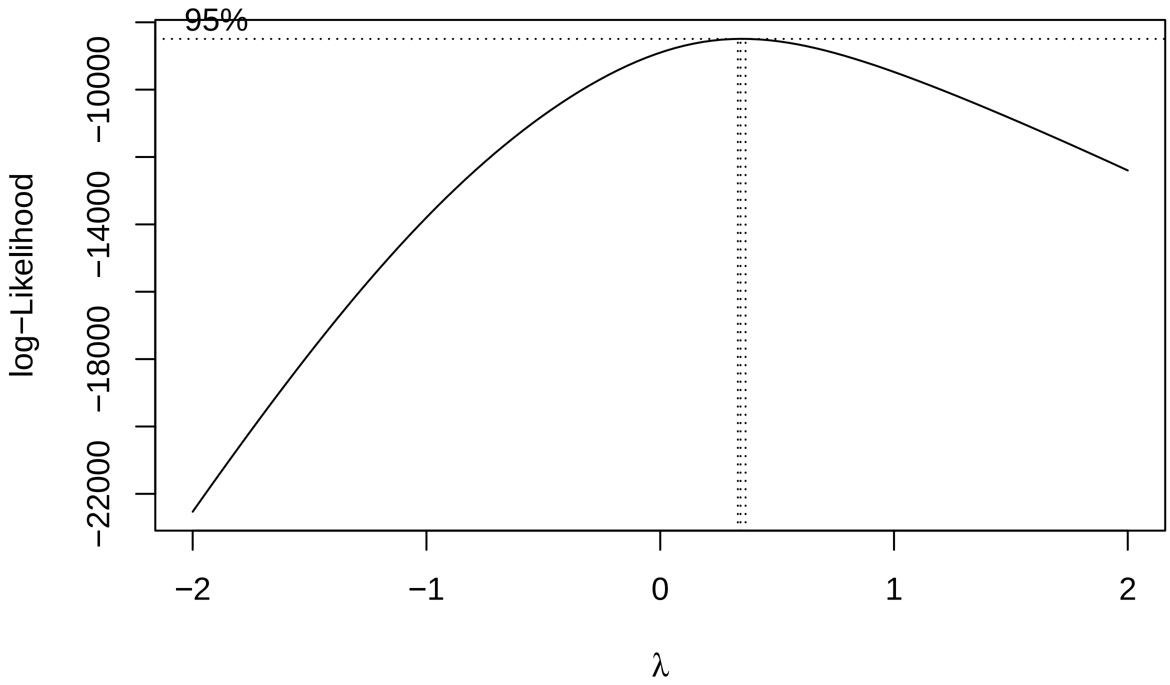
```
# Simple intercept-only model (like in tutorial)
bc_model_raw <- lm(bike_ts ~ 1)
boxcox_raw   <- MASS::boxcox(bc_model_raw, lambda = seq(-2, 2, 0.1))
```



```
(lambda_opt_raw <- boxcox_raw$x[which.max(boxcox_raw$y)])
```

```
## [1] 0.2626263
```

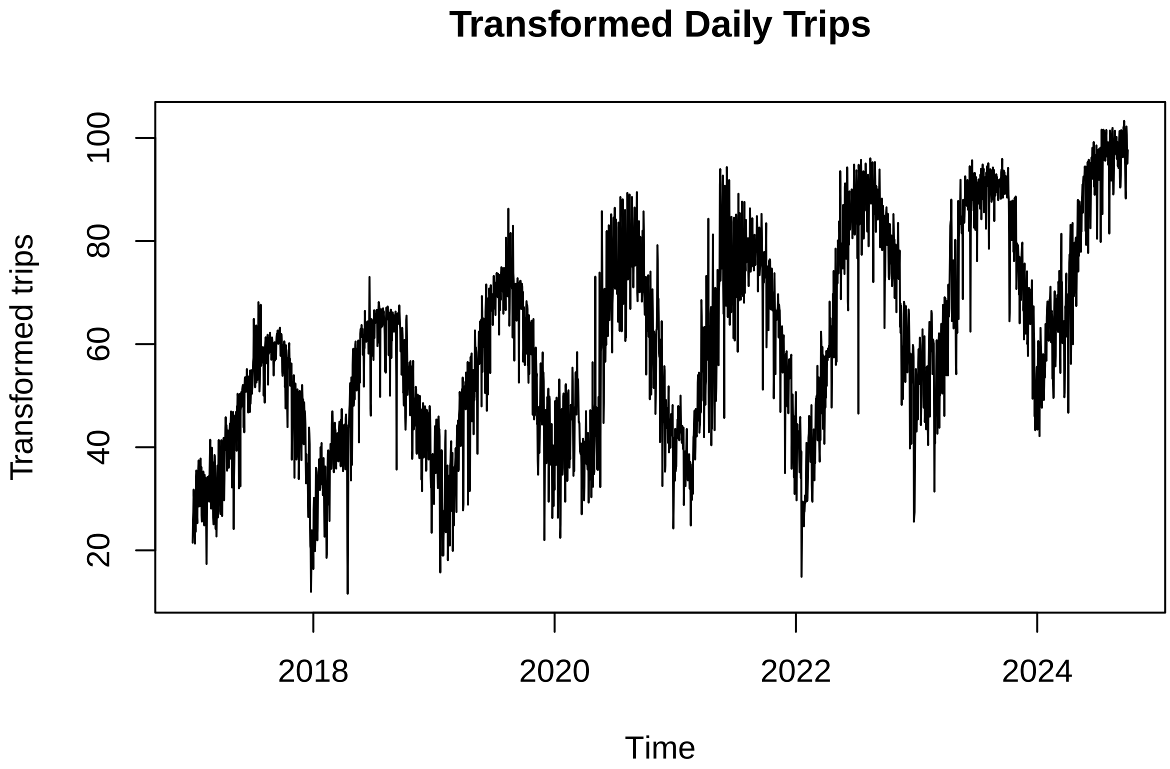
```
tim <- time(bike_ts)  # continuous time index

# Season: year and day-of-year or month; simplest is month
# Build a monthly factor from dates (instead of cycle, since this is daily)
month <- factor(format(bike$trip_date, "%m"))

reg_for_bc <- lm(bike_ts ~ tim + month)
boxcox_mod <- MASS::boxcox(reg_for_bc, lambda = seq(-2, 2, 0.1))
```



```r
(lambda_opt_mod <- boxcox_mod$x[which.max(boxcox_mod$y)])
```

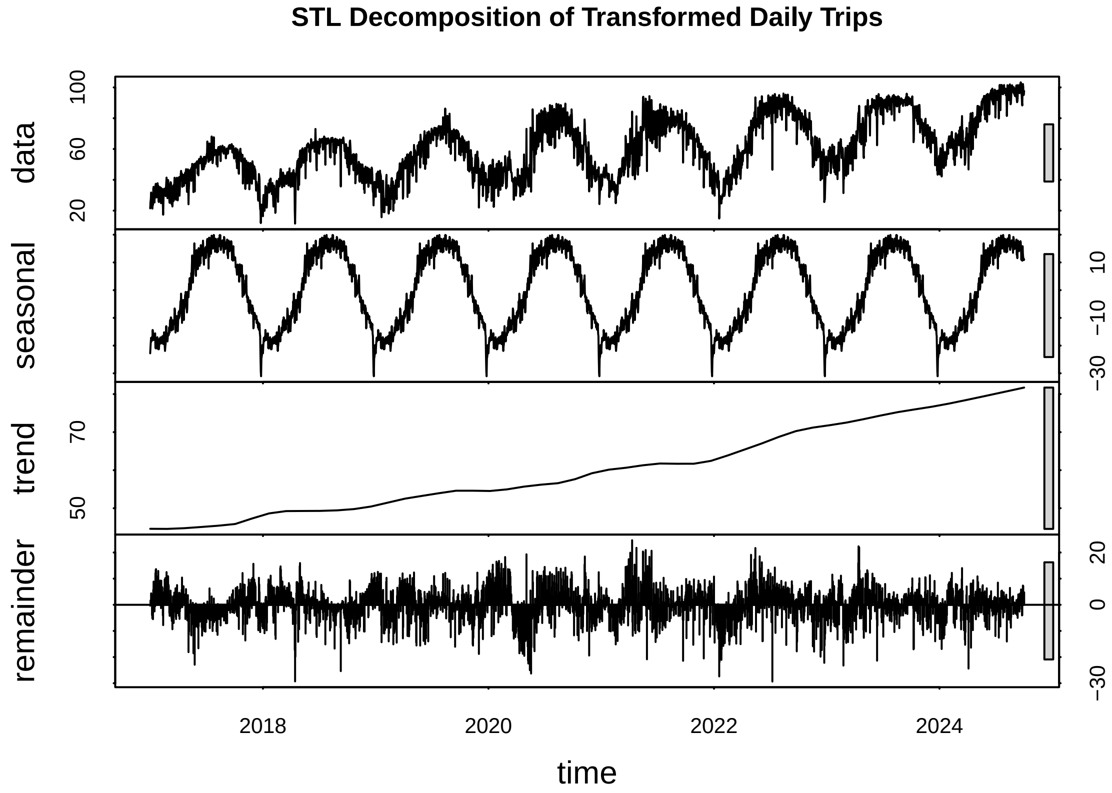
```
## [1] 0.3434343
```

```r
lam <- lambda_opt_mod   # keep this for later

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")
```



Classical Decomposition

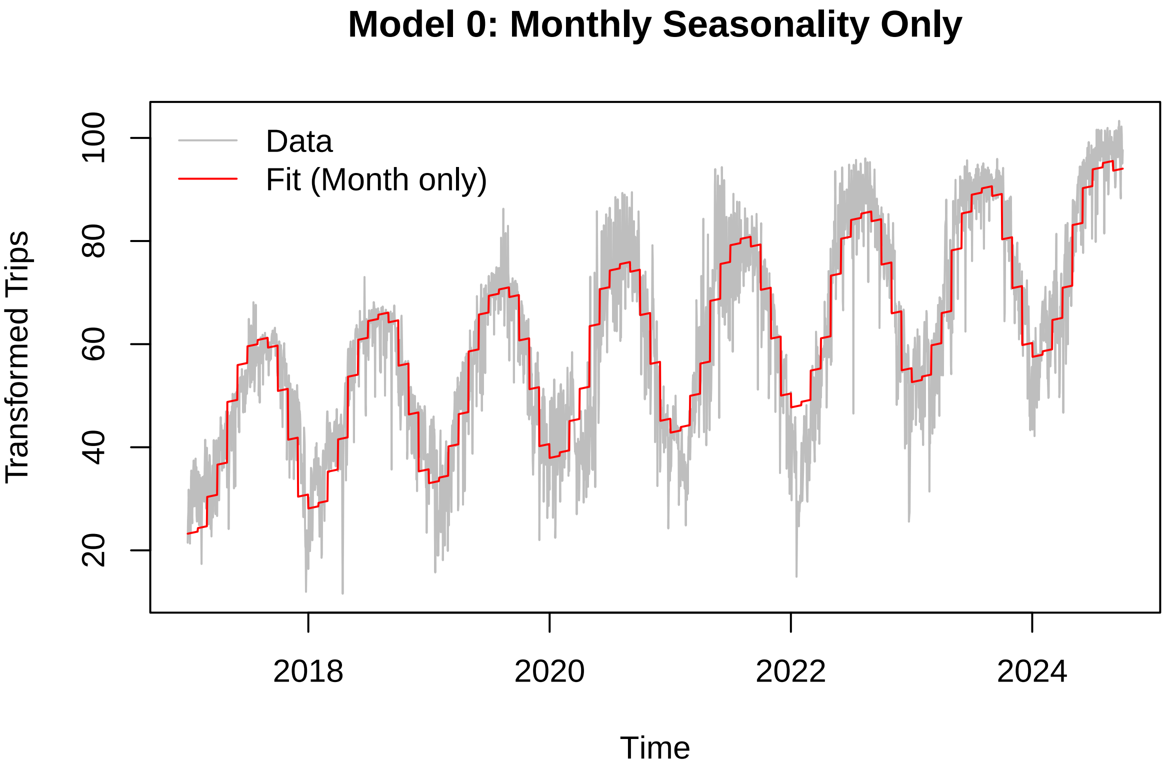
```r
Decomp_bike <- stl(y_trans, s.window = "periodic")
plot(Decomp_bike, main = "STL Decomposition of Transformed Daily Trips")
```



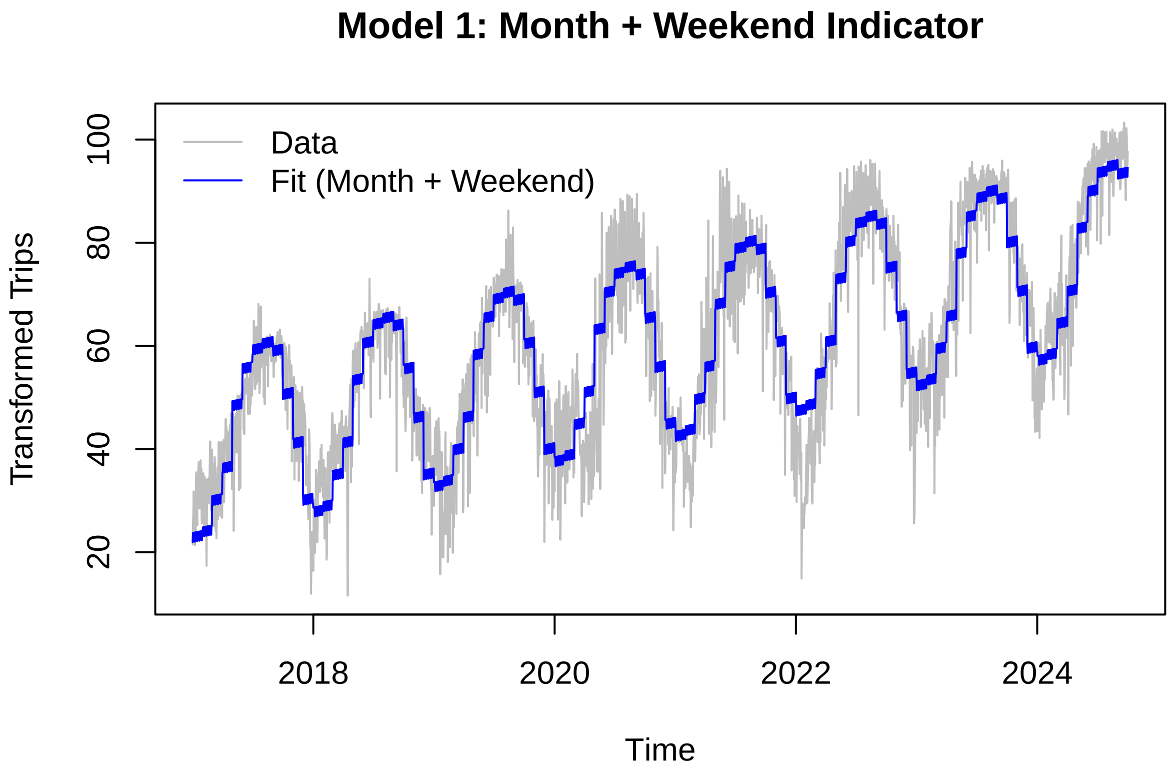
```
# Extract components if needed
bike_seasonal <- Decomp_bike$time.series[, "seasonal"]
bike_trend    <- Decomp_bike$time.series[, "trend"]
bike_remainder <- Decomp_bike$time.series[, "remainder"]
```

```
# --- Feature Engineering ---
# Time Index
tim <- time(bike_ts)

# Seasonal Factors
month <- factor(format(bike$trip_date, "%m"))
weekday_num <- as.numeric(format(bike$trip_date, "%u")) # 1=Mon, ..., 7=Sun
weekday <- factor(weekday_num)
is_weekend <- factor(ifelse(weekday_num >= 6, "Weekend", "Weekday"))

# Dataframe
model_data <- data.frame(
  y_trans = as.numeric(y_trans),
  tim = as.numeric(tim),
  month = month,
  weekday = weekday,
  is_weekend = is_weekend
)
```

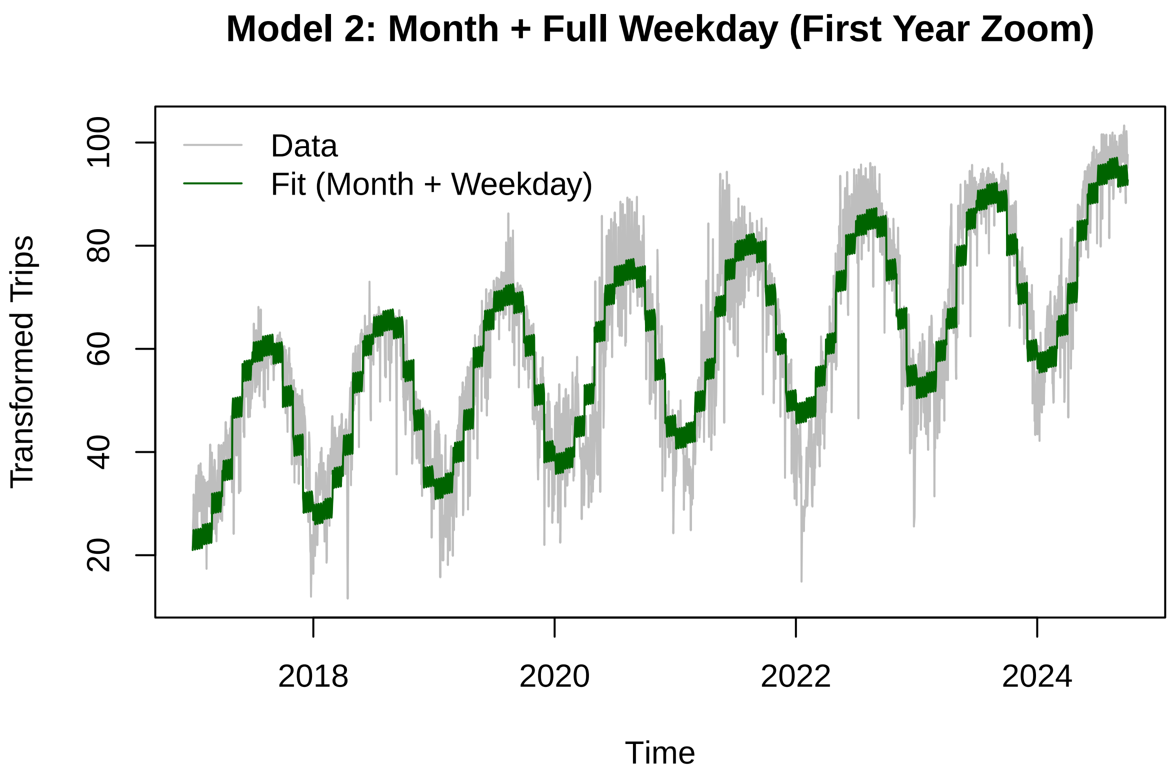
```
# --- Visualizing the Seasonality Tests ---

# Model 0: Base (Trend + Month Only)
mod_base <- lm(y_trans ~ tim + month, data = model_data)

# PLOT 0: Base Model
plot(model_data$tim, model_data$y_trans, type = "l", col = "gray",
     main = "Model 0: Monthly Seasonality Only", ylab = "Transformed Trips", xlab = "Time")
lines(model_data$tim, fitted(mod_base), col = "red", lwd = 1)
legend("topleft", legend = c("Data", "Fit (Month only)"), col = c("gray", "red"), lty = 1, bty = "n")
```



```r
# Model 1: Weekend (Trend + Month + Weekend Indicator)
mod_weekend <- lm(y_trans ~ tim + month + is_weekend, data = model_data)

# PLOT 1: Weekend Model
plot(model_data$tim, model_data$y_trans, type = "l", col = "gray",
     main = "Model 1: Month + Weekend Indicator", ylab = "Transformed Trips", xlab = "Time")
lines(model_data$tim, fitted(mod_weekend), col = "blue", lwd = 1)
legend("topleft", legend = c("Data", "Fit (Month + Weekend)"), col = c("gray", "blue"), lty = 1, bty =
```



```
# Model 2: Full Weekday (Trend + Month + Full Weekday)
mod_weekday <- lm(y_trans ~ tim + month + weekday, data = model_data)

# PLOT 2: Weekday Model
# We zoom in on a shorter window (e.g., first year) for this plot
# because the daily wiggle is hard to see on the full plot
plot(model_data$tim, model_data$y_trans, type = "l", col = "gray",
     main = "Model 2: Month + Full Weekday (First Year Zoom)", ylab = "Transformed Trips", xlab = "Time
lines(model_data$tim, fitted(mod_weekday), col = "darkgreen", lwd = 1)
legend("topleft", legend = c("Data", "Fit (Month + Weekday)"), col = c("gray", "darkgreen"), lty = 1, b
```



```
# --- Hypothesis Testing ---
# Now you perform the tests to confirm what you see in the plots
anova(mod_base, mod_weekend)
```

```
## Analysis of Variance Table
##
## Model 1: y_trans ~ tim + month
## Model 2: y_trans ~ tim + month + is_weekend
##   Res.Df    RSS Df Sum of Sq      F    Pr(>F)
## 1   2817 172833
## 2   2816 171030  1    1803.5 29.694 5.496e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
anova(mod_weekend, mod_weekday)
```

```
## Analysis of Variance Table
##
## Model 1: y_trans ~ tim + month + is_weekend
## Model 2: y_trans ~ tim + month + weekday
##   Res.Df    RSS Df Sum of Sq      F    Pr(>F)
## 1   2816 171030
## 2   2811 168164  5    2865.4 9.5794 4.453e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

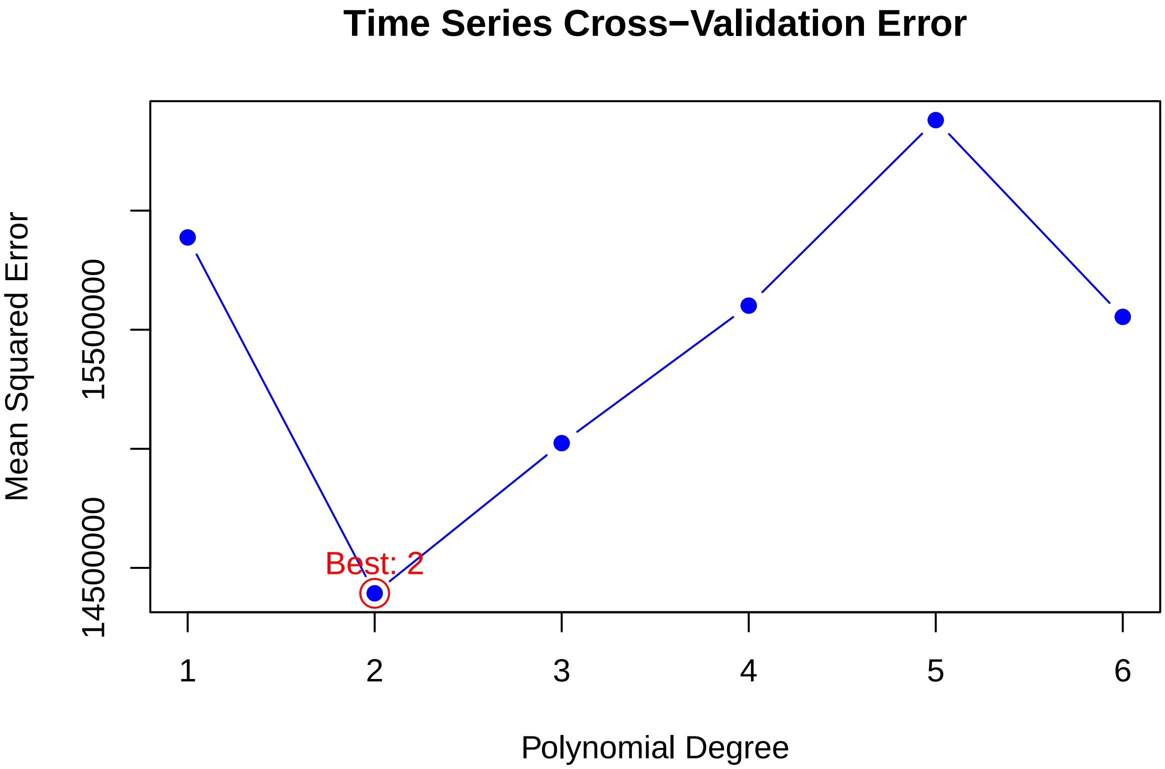
```
AIC(mod_base, mod_weekend, mod_weekday)
```

```
##             df      AIC
## mod_base    14 19696.29
## mod_weekend 15 19668.61
## mod_weekday 20 19630.79
```

```r
# --- Manual Time Series Cross-Validation ---

# 1. Setup
n_total <- length(bike_ts)
min_train <- 365 * 2  # Start training after 2 years of data
horizon <- 1          # One-step ahead forecast

# We will test degrees 1 to 6
degrees <- 1:6
cv_mse <- numeric(length(degrees))

# Time index and Monthly dummies for the WHOLE dataset first
# (It's safe to create predictors beforehand as they are deterministic)
time_idx <- 1:n_total
month_fac <- factor(cycle(bike_ts)) # 1 to 12

# Prepare the master dataframe
data_full <- data.frame(y = as.numeric(bike_ts), t = time_idx, month = month_fac)

# Loop through degrees
for (deg in degrees) {

  errors <- c()

  # 2. Rolling Loop
  # We slide the cutoff "i" from min_train to (n_total - 1)
  # This might be slow if n is large. For 2000 points it's fine.
  # To speed up, you can step by 30 days (monthly rolling) instead of 1 day.

  for (i in seq(min_train, n_total - 1, by = 30)) { # Jumping by 30 days for speed

    # Train set: 1 to i
    train_set <- data_full[1:i, ]

    # Test set: i+1
    test_set <- data_full[(i + 1), , drop = FALSE] # drop=FALSE keeps it a dataframe

    # Fit model on Train
    # Note: raw=TRUE is critical here so the coefficients mean something stable
    fit <- lm(y ~ poly(t, deg, raw = TRUE) + month, data = train_set)

    # Predict on Test
    pred <- predict(fit, newdata = test_set)

    # Calculate Squared Error
    err <- (test_set$y - pred)^2
    errors <- c(errors, err)
  }

  # Average MSE for this degree
  cv_mse[deg] <- mean(errors)
  print(paste("Degree", deg, "MSE:", round(cv_mse[deg], 4)))
}
```

```
## [1] "Degree 1 MSE: 15887410.39"
## [1] "Degree 2 MSE: 14393271.5257"
## [1] "Degree 3 MSE: 15023753.3796"
## [1] "Degree 4 MSE: 15601016.1575"
## [1] "Degree 5 MSE: 16379940.025"
## [1] "Degree 6 MSE: 15553902.3576"
```

```
# 3. Visualize Results
results_ts_cv <- data.frame(Degree = degrees, MSE = cv_mse)

plot(results_ts_cv$Degree, results_ts_cv$MSE, type = "b", col = "blue", pch = 19,
     main = "Time Series Cross-Validation Error",
     xlab = "Polynomial Degree", ylab = "Mean Squared Error")

best_d <- which.min(cv_mse)
points(best_d, cv_mse[best_d], col = "red", cex = 2, pch = 1)
text(best_d, cv_mse[best_d], paste("Best:", best_d), pos = 3, col = "red")
```

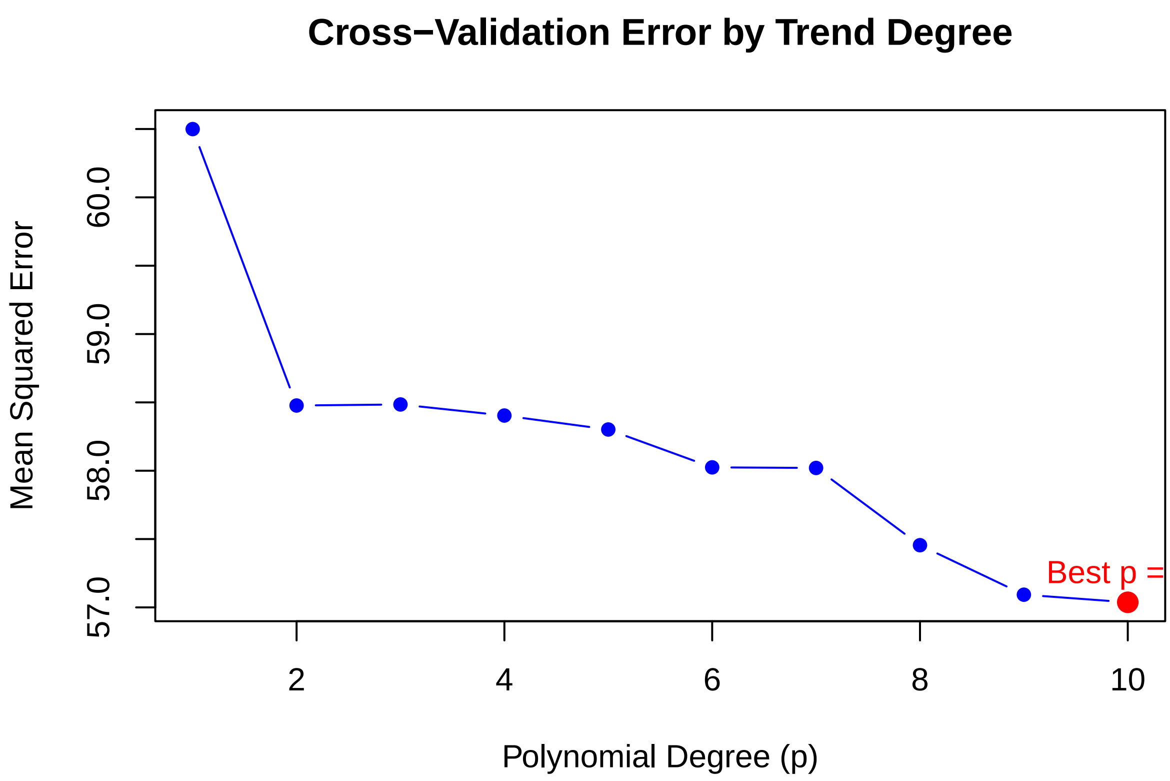


```
# --- Define the range of degrees to test ---
max_degree <- 10 # You can go higher if you want, but 6-8 is usually enough
output <- data.frame(Degree = integer(), AdjR2 = double(), AIC = double(), BIC = double(), CV_MSE = dou

# --- Setup for Cross-Validation ---
set.seed(443) # For reproducibility
n <- nrow(model_data)
k_folds <- 5
# Create random fold assignments
folds <- sample(rep(1:k_folds, length.out = n))
```

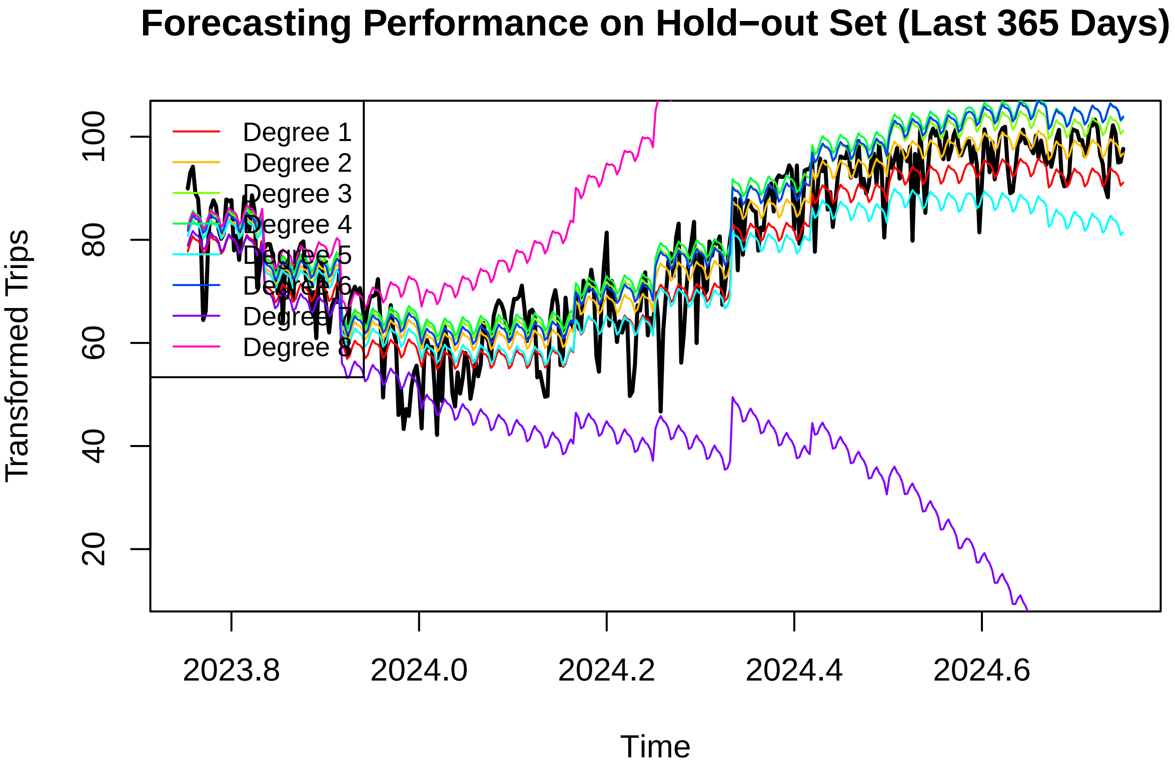
```r
for (p in 1:max_degree) {

  # 1. Fit model on FULL data to get AIC/BIC/AdjR2
  full_mod <- lm(y_trans ~ poly(tim, p, raw = FALSE) + month + weekday, data = model_data)

  # 2. Cross-Validation Loop
  cv_errors <- numeric(k_folds)
  for (k in 1:k_folds) {
    # Split data
    test_idx <- which(folds == k)
    train_data <- model_data[-test_idx, ]
    test_data  <- model_data[test_idx, ]

    # Fit on Train
    train_mod <- lm(y_trans ~ poly(tim, p, raw = FALSE) + month + weekday, data = train_data)

    # Predict on Test
    pred_test <- predict(train_mod, newdata = test_data)

    # Calc MSE
    cv_errors[k] <- mean((test_data$y_trans - pred_test)^2)
  }

  # 3. Store Results
  output[p, ] <- c(p,
                   summary(full_mod)$adj.r.squared,
                   AIC(full_mod),
                   BIC(full_mod),
                   mean(cv_errors))
}

# --- View the Selection Table ---
print(round(output, 4))
```

```
##    Degree  AdjR2      AIC      BIC  CV_MSE
## 1       1 0.8394 19630.79 19749.75 60.4994
## 2       2 0.8448 19535.02 19659.93 58.4771
## 3       3 0.8448 19536.66 19667.51 58.4847
## 4       4 0.8449 19534.22 19671.02 58.4037
## 5       5 0.8454 19527.25 19670.00 58.3015
## 6       6 0.8461 19514.63 19663.33 58.0245
## 7       7 0.8462 19514.96 19669.61 58.0203
## 8       8 0.8476 19488.67 19649.27 57.4550
## 9       9 0.8487 19470.42 19636.96 57.0930
## 10     10 0.8488 19469.74 19642.23 57.0373
```

```r
# --- Plot CV Error to find the minimum ---
plot(output$Degree, output$CV_MSE, type = 'b', pch = 16, col = "blue",
     main = "Cross-Validation Error by Trend Degree",
     xlab = "Polynomial Degree (p)", ylab = "Mean Squared Error")
# Mark the best one
best_p <- which.min(output$CV_MSE)
points(best_p, output$CV_MSE[best_p], col = "red", pch = 16, cex = 1.5)
text(best_p, output$CV_MSE[best_p], labels = paste("Best p =", best_p), pos = 3, col = "red")
```



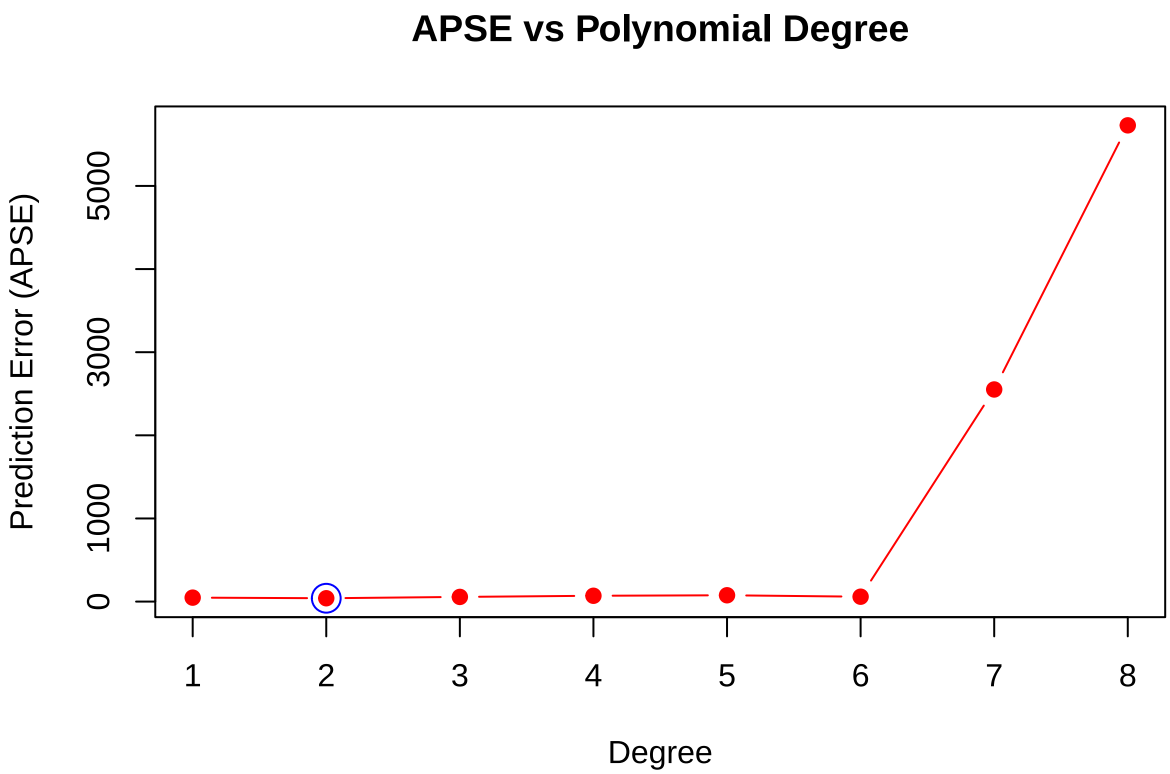
```
# --- APSE with Temporal Split (The "Honest" Test) ---

# 1. Split Data into Training (Past) and Test (Recent Future)
# Let's hold out the last 365 days as our "Test Set"
n_total <- nrow(model_data)
n_test <- 365
n_train <- n_total - n_test

train_data <- model_data[1:n_train, ]
test_data  <- model_data[(n_train + 1):n_total, ]

# 2. Loop through degrees and check forecasting error
max_degree <- 8 # Feel free to check up to 8 or 10
apse_results <- numeric(max_degree)

par(mfrow = c(1,1))
# Plot the test data to set up the canvas
plot(test_data$tim, test_data$y_trans, type = "l", col = "black", lwd = 2,
     main = "Forecasting Performance on Hold-out Set (Last 365 Days)",
     xlab = "Time", ylab = "Transformed Trips",
     ylim = range(model_data$y_trans))

colors <- rainbow(max_degree)

for (p in 1:max_degree) {
  # Fit ONLY on Training Data
  mod <- lm(y_trans ~ poly(tim, p, raw = FALSE) + month + weekday, data = train_data)
  # Predict on Test Data
  pred <- predict(mod, newdata = test_data)

  # Calculate APSE (Mean Squared Error on the Test Set)
  apse_results[p] <- mean((test_data$y_trans - pred)^2)

  # Add line to plot to visualize the "Explosion"
  lines(test_data$tim, pred, col = colors[p], lwd = 1)
}

legend("topleft", legend = paste("Degree", 1:max_degree), col = colors, lty = 1, cex = 0.8)
```



```
# 3. Print the Results
results_table <- data.frame(Degree = 1:max_degree, APSE = apse_results)
print(results_table)
```

```
##   Degree       APSE
## 1      1   47.62278
## 2      2   40.26776
## 3      3   56.12507
## 4      4   69.93335
## 5      5   76.13875
## 6      6   58.66346
## 7      7 2551.62090
## 8      8 5728.81653
```

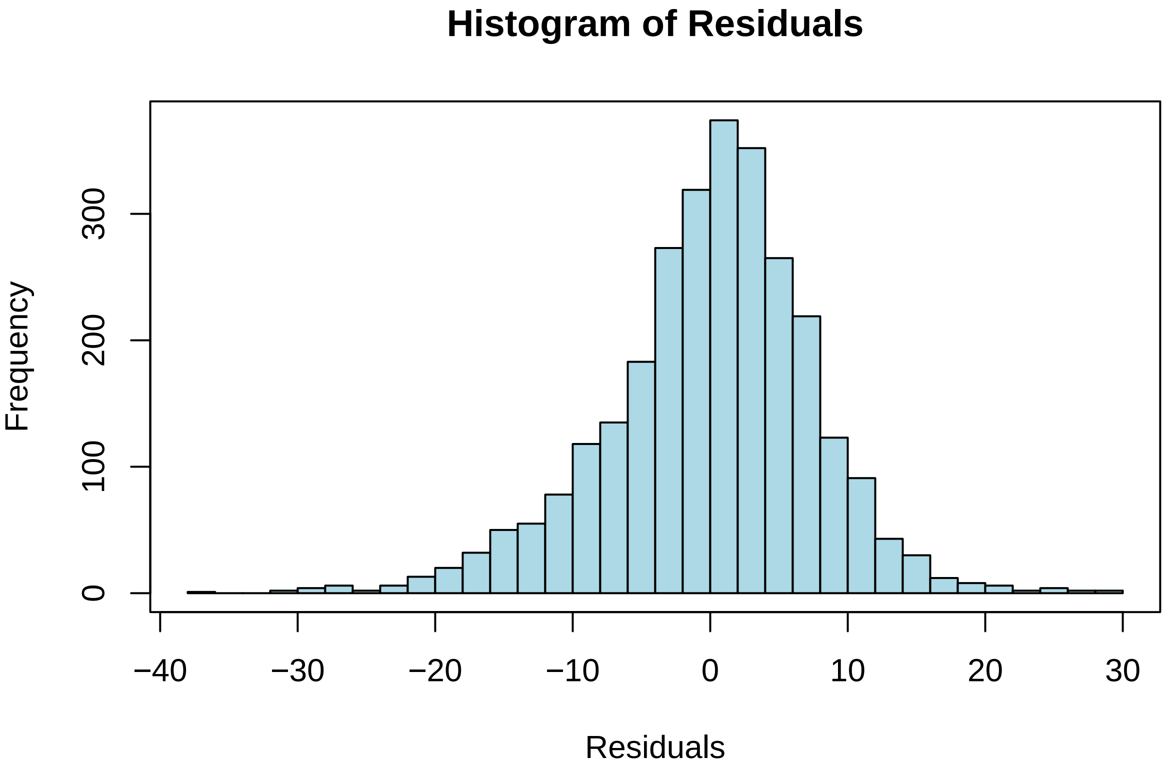
```
# Plot APSE scores
plot(results_table$Degree, results_table$APSE, type = "b", pch = 19, col = "red",
     main = "APSE vs Polynomial Degree", xlab = "Degree", ylab = "Prediction Error (APSE)")
points(which.min(apse_results), min(apse_results), cex = 2, col = "blue")
```



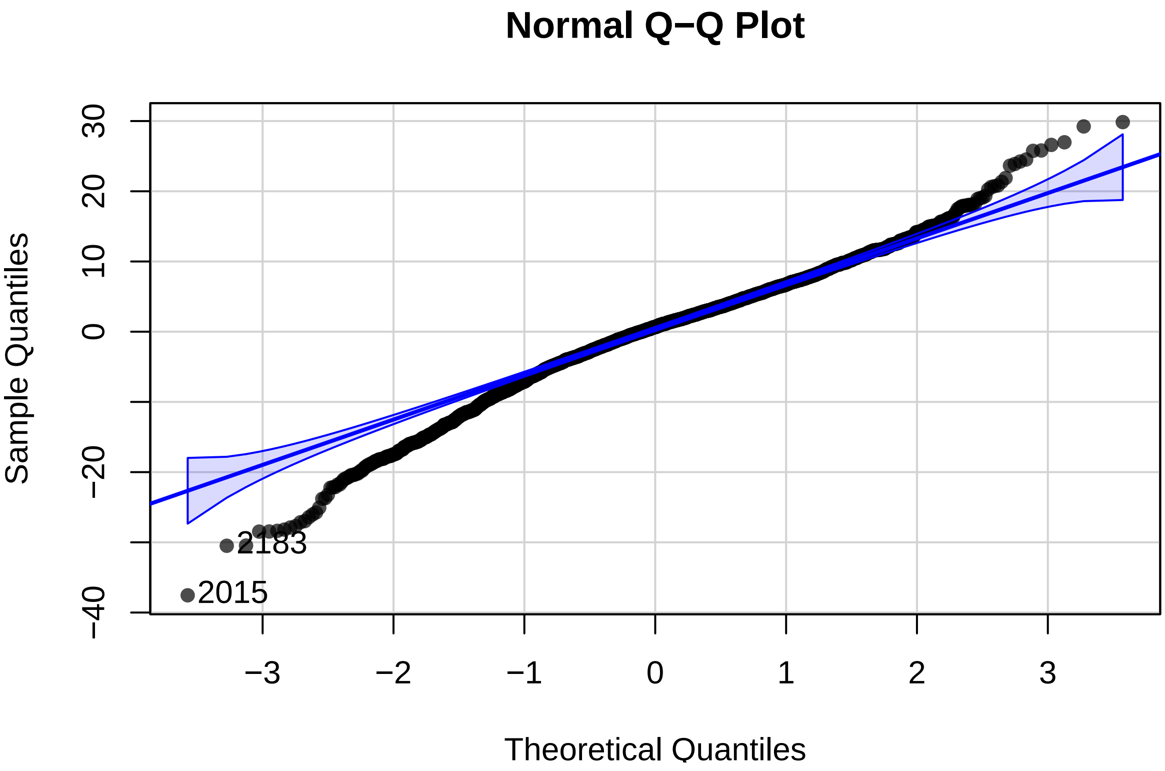
```
# --- 1. Refit the Winner (Degree 2) on Full Data ---
final_model <- lm(y_trans ~ poly(tim, 2, raw = FALSE) + month + weekday, data = model_data)
summary(final_model)
```

```
##
## Call:
## lm(formula = y_trans ~ poly(tim, 2, raw = FALSE) + month + weekday,
##     data = model_data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -37.542  -3.963   0.672   4.740  29.852
##
## Coefficients:
##                              Estimate Std. Error t value Pr(>|t|)
## (Intercept)                   40.7487     0.5953  68.455  < 2e-16 ***
## poly(tim, 2, raw = FALSE)1 583.3739     7.6311  76.447  < 2e-16 ***
## poly(tim, 2, raw = FALSE)2  76.1305     7.6601   9.939  < 2e-16 ***
## month02                        0.6597     0.6993   0.943 0.345588
## month03                        6.3322     0.6829   9.272  < 2e-16 ***
## month04                       12.2275     0.6887  17.755  < 2e-16 ***
## month05                       23.9512     0.6832  35.060  < 2e-16 ***
## month06                       30.6940     0.6890  44.546  < 2e-16 ***
## month07                       33.9593     0.6836  49.679  < 2e-16 ***
## month08                       34.7098     0.6839  50.755  < 2e-16 ***
## month09                       32.8345     0.6898  47.598  < 2e-16 ***
## month10                       24.4784     0.7083  34.559  < 2e-16 ***
## month11                       14.5496     0.7147  20.357  < 2e-16 ***
## month12                        3.1326     0.7087   4.420 1.02e-05 ***
## weekday2                       1.8325     0.5347   3.427 0.000618 ***
## weekday3                      2.9754     0.5347   5.565 2.87e-08 ***
## weekday4                      2.5131     0.5347   4.700 2.73e-06 ***
## weekday5                      2.0453     0.5347   3.825 0.000134 ***
## weekday6                      1.0950     0.5347   2.048 0.040666 *
## weekday7                     -0.8880     0.5343  -1.662 0.096634 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 7.603 on 2810 degrees of freedom
## Multiple R-squared:  0.8458, Adjusted R-squared:  0.8448
## F-statistic: 811.4 on 19 and 2810 DF,  p-value: < 2.2e-16
```

```
# --- 2. Diagnostic Plots (Generated One by One) ---

# Plot A: Histogram of Residuals
# Checks if the noise is roughly Bell-shaped
hist(final_model$residuals,
     breaks = 30,
     col = "lightblue",
     main = "Histogram of Residuals",
     xlab = "Residuals")
box() # Adds a nice border
```

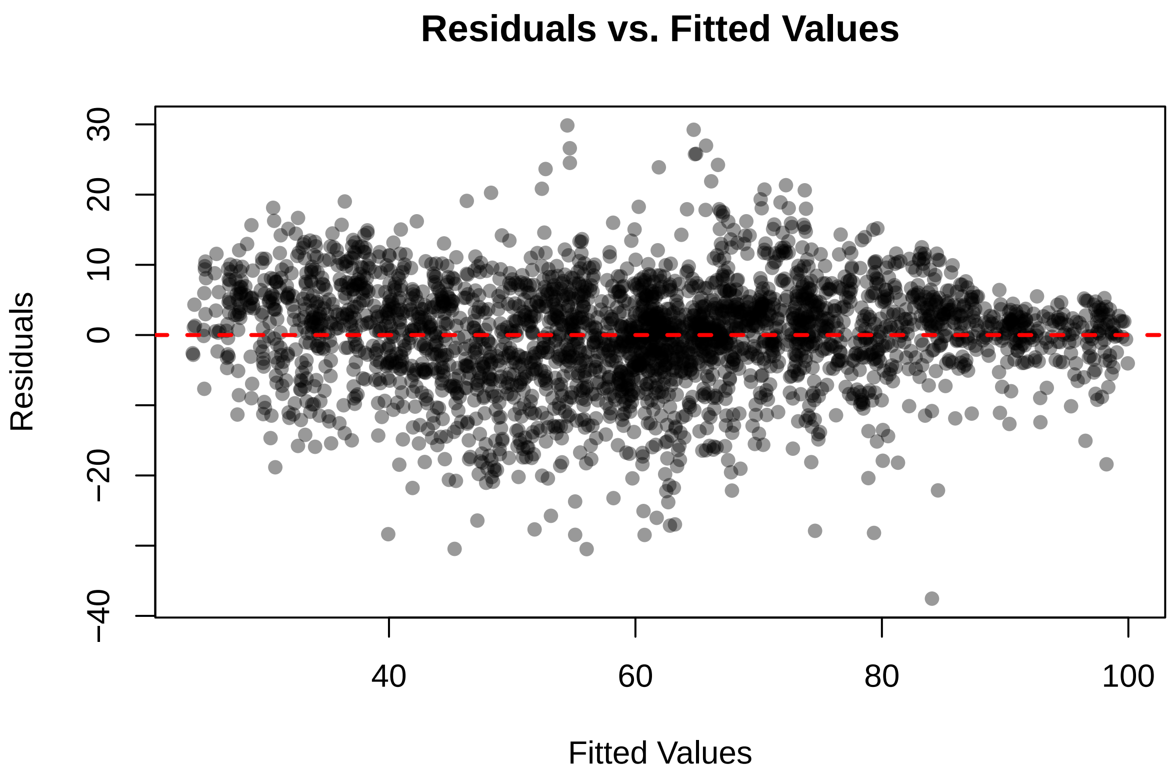


```
# Plot B: Q-Q Plot
# Checks for Normality (points should hug the blue line)
car::qqPlot(final_model$residuals,
            pch = 16,
            col = adjustcolor("black", 0.7),
            main = "Normal Q-Q Plot",
            xlab = "Theoretical Quantiles",
            ylab = "Sample Quantiles")
```

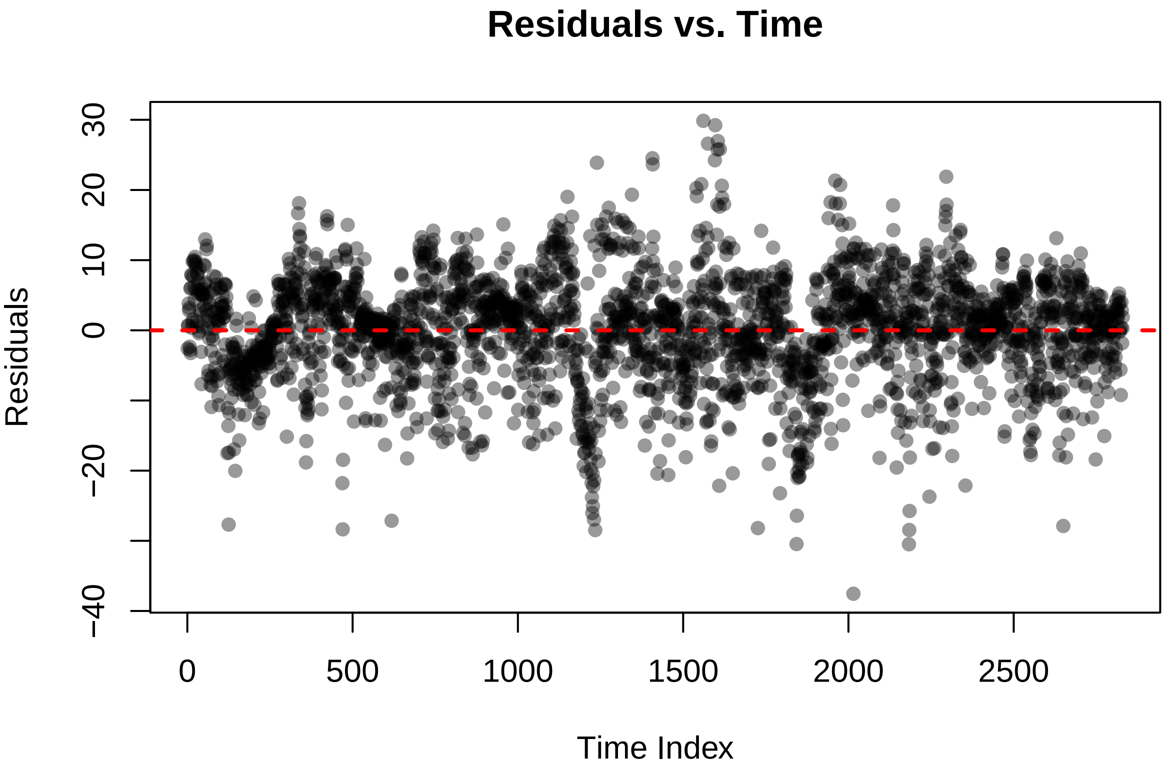


```
## [1] 2015 2183
```

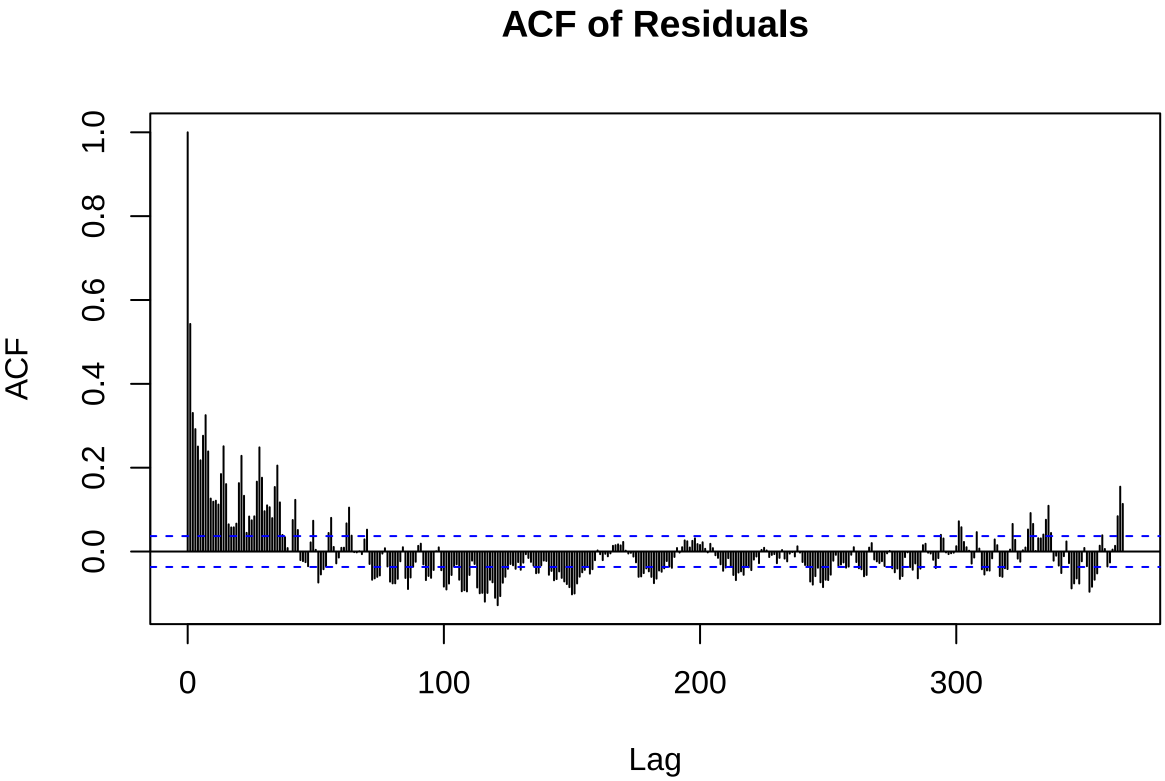
```r
# Plot C: Fitted Values vs. Residuals
# Checks for Homoscedasticity (spread should be constant, no funnel shape)
plot(fitted(final_model), residuals(final_model),
     pch = 16,
     col = adjustcolor("black", 0.4),
     main = "Residuals vs. Fitted Values",
     xlab = "Fitted Values",
     ylab = "Residuals")
abline(h = 0, lty = 2, col = "red", lwd = 2)
```



```
# Plot D: Residuals vs. Time
# Checks if variance changes over time or if there are bursts of volatility
plot(residuals(final_model),
     pch = 16,
     col = adjustcolor("black", 0.4),
     main = "Residuals vs. Time",
     xlab = "Time Index",
     ylab = "Residuals")
abline(h = 0, lty = 2, col = "red", lwd = 2)
```



```
# Plot E: ACF of Residuals
# Checks for Independence (bars should stay inside blue lines)
acf(residuals(final_model),
    lag.max = 365, # Check up to a year of lags to see long-term correlation
    main = "ACF of Residuals")
```



```r
# --- 3. Statistical Tests ---
print("--- Shapiro-Wilk Test for Normality ---")
```

```
## [1] "--- Shapiro-Wilk Test for Normality ---"
```

```r
# Limit to 5000 samples because shapiro.test() has a hard limit
if(length(residuals(final_model)) > 5000) {
  print(shapiro.test(residuals(final_model)[1:5000]))
} else {
  print(shapiro.test(residuals(final_model)))
}
```

```
##
##  Shapiro-Wilk normality test
##
## data:  residuals(final_model)
## W = 0.98126, p-value < 2.2e-16
```

```r
print("--- Kolmogorov-Smirnov Test for Normality ---")
```

```
## [1] "--- Kolmogorov-Smirnov Test for Normality ---"
```

```r
print(ks.test(scale(residuals(final_model)), "pnorm"))
```

```
##
##  Asymptotic one-sample Kolmogorov-Smirnov test
##
## data:  scale(residuals(final_model))
## D = 0.052341, p-value = 3.689e-07
## alternative hypothesis: two-sided
```

```r
print("--- Runs Test for Randomness ---")
```

```
## [1] "--- Runs Test for Randomness ---"
```

```r
# Checks if residuals have random sign flips
print(randtests::runs.test(residuals(final_model)))
```

```
##
##  Runs Test
##
## data:  residuals(final_model)
## statistic = -22.561, runs = 816, n1 = 1415, n2 = 1415, n = 2830,
## p-value < 2.2e-16
## alternative hypothesis: nonrandomness
```