Linear Regression on a Transformed Time Variable

Table of contents

To illustrate linear regression on a transformed time variable we fit a quadratic polynomial.

To reproduce the results, it is necessary to prepare the data set, plot base, and training and test data sets, as outlined in the “Data Preparation” section.

## 1 Preparation

### 1.1 Loading Required Packages and Data

Load the necessary packages, data sets, and other supporting files. Each element serves a specific purpose:

* **tidyverse**: For data manipulation and visualisation.
* **lme4 and lmerTest**: To fit and analyse mixed-effects models.
* **caret**: To compute model performance indices.
* **plot\_base**: A pre-configured ggplot object for visualisation.
* **Training and Test Data sets**: Required for cross-validation.

# Load necessary packages
library(tidyverse)
library(lme4)
library(lmerTest)
library(caret)

# Load the data set
load("data/wido.rdata")

# Load the pre-configured plot base
plot\_base <- readRDS("objects/plot\_base.rds")

# Load training and test datasets for cross-validation
training\_datasets <- readRDS("objects/training\_datasets.rds")
test\_datasets <- readRDS("objects/test\_datasets.rds")

### 1.2 Applying an Orthogonal Polynomial

To avoid multicollinearity arising from using two terms of time (a linear and a quadratic term), we use an orthogonal polynomial. This ensures that the linear and quadratic time terms are uncorrelated. The poly() function generates two orthogonal terms: the linear and the quadratic components of time, stored in poly\_time.

# Apply orthogonal polynomial transformation to the time variable
wido$poly\_time <- poly(wido$mnths, 2)

## 2 Analysis

### 2.1 Fitting the Model

Fit the linear mixed-effects model using the transformed time variable (poly\_time). This model includes both fixed effects for the linear and quadratic time terms and random effects for these terms to account for person-specific trajectories.

# Fit the linear mixed-effects model
lin <- lmer(
 lifesatisfaction ~ poly\_time[,1] + poly\_time[,2] +
 (poly\_time[,1] + poly\_time[,2] | id),
 data = wido
)

# Display the summary of the model
summary(lin)

Linear mixed model fit by REML. t-tests use Satterthwaite's method [
lmerModLmerTest]
Formula: lifesatisfaction ~ poly\_time[, 1] + poly\_time[, 2] + (poly\_time[,
 1] + poly\_time[, 2] | id)
 Data: wido

REML criterion at convergence: 5512.7

Scaled residuals:
 Min 1Q Median 3Q Max
-5.4176 -0.4947 0.0778 0.5705 3.3371

Random effects:
 Groups Name Variance Std.Dev. Corr
 id (Intercept) 0.6131 0.7830
 poly\_time[, 1] 442.7048 21.0406 0.15
 poly\_time[, 2] 66.0257 8.1256 -0.23 0.11
 Residual 0.4358 0.6601
Number of obs: 2322, groups: id, 208

Fixed effects:
 Estimate Std. Error df t value Pr(>|t|)
(Intercept) 4.95087 0.05902 207.34316 83.887 < 2e-16 \*\*\*
poly\_time[, 1] -8.94865 1.89670 102.63469 -4.718 7.54e-06 \*\*\*
poly\_time[, 2] 5.93670 1.46277 69.70255 4.059 0.000127 \*\*\*
---
Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

Correlation of Fixed Effects:
 (Intr) p\_[,1]
poly\_tm[,1] 0.109
poly\_tm[,2] 0.148 0.086

# Compute confidence intervals for the model parameters
round(confint(lin), 2)

 2.5 % 97.5 %
.sig01 0.70 0.87
.sig02 -0.04 0.34
.sig03 -0.89 0.15
.sig04 17.36 24.92
.sig05 -0.84 0.70
.sig06 1.05 12.74
.sigma 0.64 0.68
(Intercept) 4.83 5.07
poly\_time[, 1] -13.08 -4.76
poly\_time[, 2] 2.49 9.37

## 3 Visualisation

### 3.1 Bootstrapping Confidence Intervals

Use bootstrapping to estimate the confidence intervals for the predicted values of the model. This provides a robust measure of uncertainty.

# For reproducibility
set.seed(123)

# Bootstrapping for confidence intervals of the predictions
boot\_results <- bootMer(
 lin,
 FUN = function(x) predict(x, newdata = wido, re.form = NA),
 nsim = 1000
)

# Extract the 95% confidence intervals from the bootstrapped results
ci <- apply(boot\_results$t, 2, quantile, probs = c(0.025, 0.975))

# Assign the lower and upper bounds to the data
wido$lower\_bound <- ci[1, ]
wido$upper\_bound <- ci[2, ]

### 3.2 Predicting Average and Individual Trajectories

Predict both the population-level (fixed effects) and individual-level (random effects) trajectories of life satisfaction.

# Predict population-level trajectories based on fixed effects
wido$lifesatisfaction\_lin\_f <- predict(lin, newdata = wido, re.form = NA)

# Predict individual-level trajectories based on random effects
wido$lifesatisfaction\_lin\_r <- predict(lin, newdata = wido, re.form = NULL, allow.new.levels = TRUE)

### 3.3 Selecting a Random Sample for Plotting

For better visualisation, select a random sample of individuals to display their individual trajectories.

# For reproducibility
set.seed(123)

# Randomly sample 50 participants
rsample\_ids <- sample(unique(wido$id), 50)

# Filter the data to include only the randomly selected participants
wido\_rsample <- wido %>%
 filter(id %in% rsample\_ids)

### 3.4 Creating the Plot

Combine all elements to create the plot, which includes individual trajectories, the population trajectory, and the confidence interval of the population trajectory.

# Create the plot using the pre-configured plot base
plot\_base +
 geom\_line(
 data = wido\_rsample,
 aes(x = mnths, y = lifesatisfaction\_lin\_r, group = id),
 color = "grey70", linewidth = 0.4
 ) +
 geom\_ribbon(
 data = wido,
 aes(x = mnths, ymin = lower\_bound, ymax = upper\_bound),
 fill = "firebrick4", alpha = 0.2
 ) +
 geom\_line(
 data = wido,
 aes(x = mnths, y = lifesatisfaction\_lin\_f),
 color = "firebrick4", linewidth = 1
 ) +
 ggtitle("Linear Regression with a Quadratic Transformation") +
 theme(plot.title = element\_text(size = 13, face = "bold"))



## 4 Model Performance

### 4.1 Evaluating the Model

Assess the model’s performance using the Bayesian Information Criterion (BIC), R-squared (R²), Mean Absolute Error (MAE), and Root Mean Squared Error (RMSE).

# Compute BIC for the fitted model
round(BIC(lin), 2)

[1] 5590.19

# Calculate R², MAE, and RMSE for the fixed effects predictions
data.frame(
 R2\_FE = round(R2(wido$lifesatisfaction\_lin\_f, wido$m\_lifesat\_per\_mnth), 2),
 MAE\_FE = round(MAE(wido$lifesatisfaction\_lin\_f, wido$m\_lifesat\_per\_mnth), 2),
 RMSE\_FE = round(RMSE(wido$lifesatisfaction\_lin\_f, wido$m\_lifesat\_per\_mnth), 2)
)

 R2\_FE MAE\_FE RMSE\_FE
1 0.14 0.33 0.43

# Calculate R², MAE, and RMSE for the random effects predictions
data.frame(
 R2\_RE = round(R2(wido$lifesatisfaction\_lin\_r, wido$lifesatisfaction), 2),
 MAE\_RE = round(MAE(wido$lifesatisfaction\_lin\_r, wido$lifesatisfaction), 2),
 RSME\_RE = round(RMSE(wido$lifesatisfaction\_lin\_r, wido$lifesatisfaction), 2)
)

 R2\_RE MAE\_RE RSME\_RE
1 0.7 0.46 0.61

### 4.2 Cross-Validation

To assess the replicability of the model, perform cross-validation using the training and test data sets. For each training data set, fit the model and compute performance metrics for the associated test data set R², MAE, and RMSE.

# Initialise vectors to store performance metrics
R2\_values <- c()
MAE\_values <- c()
RMSE\_values <- c()

# Perform cross-validation
for (i in seq\_along(training\_datasets)) {
 train\_data <- training\_datasets[[i]]
 test\_data <- test\_datasets[[i]]

 # Apply polynomial transformation to time variable
 train\_data$poly\_time <- poly(train\_data$mnths, 2)
 test\_data$poly\_time <- poly(test\_data$mnths, 2)

 # Fit the linear mixed-effects model on training data
 lin <- lmer(
 lifesatisfaction ~ poly\_time[,1] + poly\_time[,2] +
 (poly\_time[,1] + poly\_time[,2] | id),
 data = train\_data
 )

 # Make predictions on the test data
 test\_predictions <- predict(lin, newdata = test\_data, re.form = NA)

 # Compute average trajectory in the test data
 test\_data <- test\_data %>%
 group\_by(mnths) %>%
 mutate(m\_lifesat\_per\_mnth = mean(lifesatisfaction, na.rm = TRUE))

 # Calculate performance metrics
 R2\_values <- c(R2\_values, R2(test\_predictions, test\_data$m\_lifesat\_per\_mnth))
 MAE\_values <- c(MAE\_values, MAE(test\_predictions, test\_data$m\_lifesat\_per\_mnth))
 RMSE\_values <- c(RMSE\_values, RMSE(test\_predictions, test\_data$m\_lifesat\_per\_mnth))
}

# Compute average performance metrics (mean)
 average\_R2 <- mean(R2\_values)
 average\_MAE <- mean(MAE\_values)
 average\_RMSE <- mean(RMSE\_values)

# Compute average performance metrics (SD)
 sd\_R2 <- sd(R2\_values)
 sd\_MAE <- sd(MAE\_values)
 sd\_RMSE <- sd(RMSE\_values)

# Combine the mean and standard deviation into one data.frame
combined\_metrics <- data.frame(
 Metric = c("R²", "MAE", "RMSE"),
 Mean = round(c(average\_R2, average\_MAE, average\_RMSE), 2),
 SD = round(c(sd\_R2, sd\_MAE, sd\_RMSE), 2)
)

# Print the combined metrics
print(combined\_metrics)

 Metric Mean SD
1 R² 0.06 0.04
2 MAE 0.63 0.07
3 RMSE 0.83 0.10