

# Linear Regression on a Transformed Time Variable

## Table of contents

Preparation . . . . .	1
Loading Required Packages and Data . . . . .	1
Applying an Orthogonal Polynomial . . . . .	2
Analysis . . . . .	3
Fitting the Model . . . . .	3
Visualisation . . . . .	4
Bootstrapping Confidence Intervals . . . . .	4
Predicting Average and Individual Trajectories . . . . .	5
Selecting a Random Sample for Plotting . . . . .	5
Creating the Plot . . . . .	6
Model Performance . . . . .	7
Evaluating the Model . . . . .	7
Cross-Validation . . . . .	8

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.

## Preparation

### 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")
```

## 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)
```

## Analysis

### 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)
```

Computing profile confidence intervals ...

	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

---

## Visualisation

### 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, ]

```

## 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)

```

## 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)

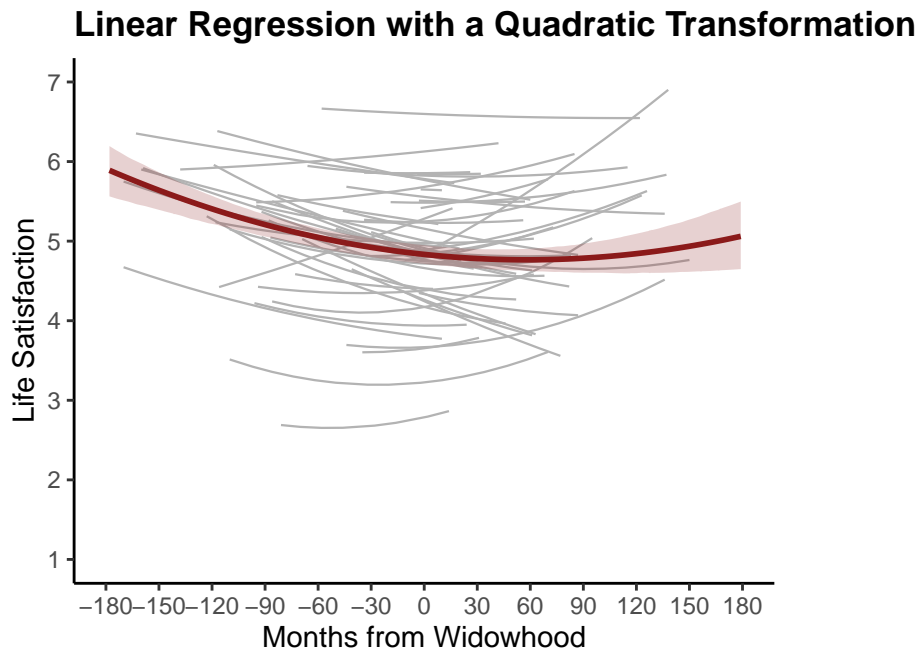
# Filter the data to include only the randomly selected participants
wido_rsample <- wido %>%
  filter(id %in% rsample_ids)

```

## 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"))
```



---

## Model Performance

### Evaluating the Model

Assess the model's performance using the Bayesian Information Criterion (BIC), R-squared ( $R^2$ ), Mean Absolute Error (MAE), and Root Mean Squared Error (RMSE).

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

```
[1] 5590.19
```

```
# Calculate  $R^2$ , 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  $R^2$ , 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

## 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  $R^2$ , 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	R <sup>2</sup>	0.06	0.04
2	MAE	0.63	0.07
3	RMSE	0.83	0.10